Escuela
Politécnica
Superior

Generación de resúmenes objetivos a partir de opiniones variadas sobre hoteles

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Alejandro Reyes Albillar
Tutor/es:
Elena Lloret Pastor

Septiembre 2018

Universitat d'Alacant
Universidad de Alicante

Generación de resúmenes objetivos a partir de opiniones variadas sobre hoteles

Autor
Alejandro Reyes Albillar

Directores
Elena Lloret Pastor
Departamento de Lenguajes y Sistemas Informáticos

Grado en Ingeniería Informática

Escuela Politécnica Superior

ALICANTE, 5 de septiembre de 2018

Preámbulo

El presente Trabajo Final de Grado (TFG) consiste en el análisis, desarrollo e implementación de una herramienta de generación de resúmenes objetivos a partir de opiniones variadas sobre hoteles.

El auge de la Web 2.0, junto con el desarrollo de nuevas maneras de interactuar con otras personas a través de internet, han ocasionado la creación del llamado "Contenido generado por el usuario", User Generated Content (UGC) en Inglés. Este hecho conlleva la creación de una inmensa cantidad de datos que es muy difícil y lento de sintetizar por una persona.

Utilizando técnicas de Procesamiento de Lenguaje Natural (PLN) intentaremos abordar este problema para poder desarrollar una solución que puedan utilizar de manera rápida y sencilla las personas.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y estímulo de mis compañeros del Grupo de Procesamiento del Lenguaje natural y Sistemas de Información (GPLSI), como Marta, Cristina, Lea, Antonio, Isa, Javi, Álvaro y muchos otros que me han ido apoyando y prestando su ayuda en las diferentes partes del desarrollo de este TFG.

Desde los primeros días que pasé en el laboratorio, hace ya un año largo, cuando empezaba mis prácticas en empresa, me fueron guiando en este gran y complejo mundo que es el PLN y las Tecnologías del Lenguaje Humano (TLH).

Tanto Álvaro como Jose Manuel fueron de muchísima ayuda con el uso y manejo del crawler Sensei, desarrollado por el GPLSI, así como Marta y Cristina, con quienes he estado trabajando desde el principio y que me han dado una grandísima cantidad de conocimiento, sobre todo del Scripting que me ha ayudado a agilizar el proceso de generación de resúmenes y tratamiento del corpus recolectado.

También agradecer a Suilán y Alex, los mejores (y únicos) cubanos que he conocido, por enseñarme y ayudarme tantísimo con python y sus librerías. Sus consejos y experiencia hicieron que el desarrollo de este TFG fuera muchísimo más rápido y preciso.

Gracias a todos ellos y a mi profesora y tutora de este TFG, Elena Lloret, he podido adentrarme en este mundillo.

El hecho de haberme facilitado acceso a un entorno de trabajo como el Laboratorio de Investigación del GPLSI es algo que no dejaré de agradecerle, ya que me ha hecho crecer como persona e iniciarme en el mundo de la investigación.

Por último agradecer a mi familia que, pese a estar preguntándome constantemente cuándo terminaré la carrera, ha estado ahí en los buenos y malos momentos, ayudando desde las sombras con dinero para las comidas y el bus y siendo un constante apoyo desde que tengo uso de razón.

Es a todos ellos a quienes dedico este trabajo.

A mis padres Pilar y Antonio,
y a mis compañeros del GPLSI
sin los cuales no habría podido llegar
hasta donde estoy ahora.

El auténtico problema no es
si las máquinas piensan,
sino si lo hacen los hombres.

Burhus Frederic Skinner.

Índice general

Índice de figuras

Índice de tablas

Índice de Listados

Lista de Acrónimos

TFG: Trabajo Final de Grado
GPLSI: Grupo de Procesamiento del Lenguaje natural y Sistemas de Información
UGC: User Generated Content
PLN: Procesamiento de Lenguaje Natural
TLH: Tecnologías del Lenguaje Humano
IA: Inteligencia Artificial
OCR: Reconocimiento Óptico de Caracteres
SRL: Semantic Role Labelling
PoS: Part of Speech
BIU: Basic Information Unit
NGD: Normalized Google Distance
SOPMI: Semantic Orientation Pointwise Mutual Information
NLTK:Natural Language Toolkit
ML: Machine Learning
LDA: Latent Dirichlet Allocation
BBDD: Bases de Datos
RNN: Redes Neuronales Recursivas
EER: Esquemas Entidad-Relación
UML: Unified Modeling Language

1. Introducción

La sociedad actual está evolucionando. La aparición de la web 2.0 y el desarrollo de la sociedad 2.0 han supuesto un enorme aumento de la cantidad de datos a disposición del usuario de a pie.

Más específicamente, si hablamos del sector de los viajes, nos encontramos con foros, redes sociales y otro tipo de plataformas que incluyen información que puede ser relevante para un usuario específico a la hora de decidir sobre su futuro viaje.

Las opiniones, valoraciones e información disponible en estos medios suponen una gran ayuda al usuario a la hora de tomar decisiones, pero suele ser difícil para este el hecho de tener que sintetizar tal cantidad de datos.

Antes de que existieran estas tecnologías, los usuarios visitaban agencias de viajes que les recomendaban destinos y hoteles populares entre sus usuarios, o preguntaban a conocidos y familiares para orientarse sobre su viaje.

Ahora toda esa información la comparten libremente los usuarios creando el conocido User Generated Content (UGC) o "Conocimiento Generado por el Usuario". Mientras que antes esta información era escasa, ahora la encontramos en exceso, a tal punto en que se necesita ayuda de plataformas para poder sintetizarla y obtener los puntos más relevantes para el usuario.

La Tabla 1.1 muestra el número de comentarios generados por los usuarios de la página TripAdvisor para algunos hoteles hasta el día 5 de Marzo de 2018:

Hotel	**Comentarios**
Luxor Hotel & Casino Las Vegas	32,007
New York Hilton Midtown	10,851
Melia Alicante	4,752
The Plaza	2,509

Tabla 1.1.: Comentarios sobre hoteles (05/03/2018)

Como podemos observar, el volumen de comentarios es considerablemente elevado para que una persona tenga que ponerse a leer y analizar cada uno de ellos de forma manual, teniendo en cuenta, además, la subjetividad que puede tener el texto con respecto a la veracidad, ya que algunos de estos comentarios pueden haber sido escritos por personal del hotel o un entorno cercano al mismo, centrándose únicamente en los aspectos positivos del lugar en vez de dar una visión general.

Por ejemplo, podemos ver que, dependiendo del tipo de viajero, algunos comentarios se contradicen para un mismo hotel:"Mala ubicación, ya que está lejos del centro del

Strip.",“Su ubicación y comunicación con otros centros de diversión como el Mandalay o el Excalibur o The Strip le hacen estratégica.”[1].

Si miramos este mismo hotel en su versión inglesa, podemos comprobar que este tipo de contradicciones se dan también para otros idiomas:“...last night no one cleaned our room 27160 we had no toilet paper or towels...”, “...The room was clean...” [2].

El hecho de encontrar opiniones contradictorias para un mismo hotel hace que sea complicado para el usuario escoger cuáles de ellas se adecúan más a su perfil o necesidades y cuales no, por lo que se debe seguir algún conjunto de criterios para clasificar estas opiniones.

Por poner un ejemplo, ¿qué ocurría si los comentarios que está viendo el usuario hubieran sido escritos específicamente para desprestigiar al hotel? En el caso de que el usuario únicamente viera opiniones negativas sobre un aspecto de un hotel, sin fijarse en las opiniones que hablan bien sobre ese aspecto específico, podrían conducirle a tomar una mala decisión sobre su viaje al haber recibido información sesgada.

Es por todo esto que la creación de una aplicación que sea capaz de resumir objetivamente esta información, mostrando tanto los aspectos positivos como negativos de cada uno de los hoteles haciendo un análisis de la contradicción, sería de ayuda para los usuarios que desean saber de forma fácil y sencilla cuál es la mejor opción de alojamiento para su viaje.

En la actualidad no existen herramientas que ofrezcan este tipo de servicio, sin embargo, hace un par de años se desarrolló una herramienta que realizaba una aproximación bastante cercana a la que se busca desarrollar en este TFG. Dicha herramienta se llama TravelSum[3] y utiliza datos de Twitter y TripAdvisor sobre hoteles y restaurantes para obtener sus resúmenes. La herramienta nos da 3 vistas de la información, una positiva, una negativa y una mixta, sobre los hoteles y restaurantes seleccionados, siendo capaz de abstraer gran parte de la información recolectada. También nos muestra un gráfico sobre la calidad de los servicios del hotel, comparándolos con la media calculada en base de los hoteles de la ciudad en la que se encuentra, así como un mapa con la localización del mismo.

TravelSum es un ejemplo de sistema de resúmenes abstractivos multigénero, ya que utiliza información de diferentes fuentes para generar los resúmenes, dejándonos ver los diferentes puntos de vista existentes en dicha información, a diferencia de los sistemas de resúmenes tradicionales.

Sin embargo, a diferencia de lo que se propone en este Trabajo Final de Grado (TFG), TravelSum no tiene en cuenta la diferencia de opinión sobre un aspecto específico, debido a que únicamente agrupa comentarios de una misma polaridad.

Esta diferencia es el hecho de que cuando una persona habla sobre un aspecto específico, existe la posibilidad de que el resto no piense igual que ella, lo que introduce sesgo en la opinión y la entrega de información a los usuarios de la herramienta.

[1] `https://www.tripadvisor.es/Hotel_Review-g45963-d111709-Reviews-Luxor_Hotel_Casino-Las_Vegas_Nevada.html`

[2] `https://www.tripadvisor.co.uk/Hotel_Review-g45963-d111709-Reviews-Luxor_Las_Vegas-Las_Vegas_Nevada.html`

[3] `http://travelsum.gplsi.es/`

El objetivo de este TFG consiste en dar un paso más allá y poder dar la información que estos medios proporcionan de manera resumida, teniendo en cuenta, además, el tipo de viajante que consulta la información de modo que pueda acercarse más al usuario y le sea de mayor utilidad la información generada.

Además de ello, este TFG tiene en cuenta que en la información que nos proporcionan los comentarios existe un sesgo o subjetividad en función de la experiencia que cada usuario haya tenido en ese hotel, de modo que analizamos dicho sesgo para aportar al usuario final una vista más global sobre el hotel, en vez de una simple selección de frases como realizan otros sistemas. De esta manera se puede obtener un resumen más objetivo en lo que a opiniones respecta.

2. Marco Teórico

El siguiente TFG se enmarca dentro del ámbito del Procesamiento de Lenguaje Natural (PLN), más concretamente en el ámbito de la generación de resúmenes de forma automática, por lo que a continuación procederemos a explicar qué son estos conceptos y en qué se basan para entender más claramente en qué consiste este trabajo.

2.1. ¿Qué es el Procesamiento de Lenguaje Natural (PLN)?

El PLN es el campo de la informática relacionado con la Inteligencia Artificial (IA) que se centra en la interacción Persona-Máquina, es decir, entre los ordenadores y el lenguaje natural humano[1].

Más concretamente, se trata de la parte de la informática que trata de técnicas computacionales que procesan lenguaje humano escrito o hablado [Jurafsky, 2000].

El lenguaje humano es algo muy complejo y que cambia dependiendo de la zona geográfica, la época e incluso el estrato social de una persona. Por culpa de esto, el procesamiento de este es complejo, debido a un conjunto de problemas que mencionamos a continuación.

2.1.1. Problemas existentes que dificultan el procesamiento computacional del lenguaje

Al tratar con un conjunto de información en un idioma concreto, nos encontramos una serie de problemas[1], listados a continuación, que nacen a raíz de las distintas interpretaciones que se le puede dar a dicha información según el idioma en el que se encuentre expresada:

- **Ambigüedad:** Entendemos por ambigüedad en el lenguaje el hecho de que una palabra, o expresión, pueda ser interpretada de distinta manera dependiendo del contexto en el que se encuentre, lo cual supone un problema a la hora de procesar la información. De ello se puede diferenciar entre 4 tipos distintos de ambigüedad:
  - A **nivel léxico** una misma palabra puede tener varios significados léxicos, debiendo deducir el apropiado basándose en el contexto oracional o conocimiento básico. Por ejemplo, en el caso de la palabra "casa" puede referirse al sustantivo referente a un hogar, o la tercera persona del singular del verbo "casar".

[1] `https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales`

- A **nivel referencial**, la resolución de anáforas y catáforas implica determinar la entidad lingüística previa o posterior a que hacen referencia, respectivamente. Por ejemplo, la frase "Vinieron Pedro y Juan: éste estaba enfadado y aquél contento." es un ejemplo de anáfora, haciendo referencia a Pedro y Juan con las palabras éste y aquel sin haber especificado quien es quien.
- A **nivel estructural/semántico**, se requiere de la semántica para desambiguar la dependencia de los sintagmas preposicionales que conducen a la construcción de distintos árboles sintácticos. Por ejemplo, la frase "Creó un resumen de varias páginas" puede interpretarse de distinta manera dependiendo de qué árbol semántico se genere. Podría ser que el resumen fuera generado a partir de varias páginas o que el resumen resultado constara de varias páginas. Del mismo modo, se utiliza la semántica para diferenciar los distintos significados de una palabra dentro de una misma categoría léxica. Por ejemplo, en el caso de la palabra "banco" nos encontramos con que para el Español tiene 10 significados distintos[2] de los cuales algunos de ellos son: el "banco" de peces, el "banco" mobiliario (para sentarse o trabajar sobre él) y el "banco" entidad financiera.
- A **nivel pragmático/sintáctico**, una oración, a menudo, no significa lo que realmente se está diciendo. Elementos como la ironía tienen un papel importante en la interpretación del mensaje. Este tipo de ambigüedad se encuentra sobre todo en el lenguaje hablado, ya que depende mucho del contexto en el que se encuentre y la manera en la que se expresa por el hablante. Por ejemplo, la frase "si... seguro..." se utiliza en una conversación para indicar un cierto grado de incredulidad frente a lo que se ha dicho anteriormente (e.g.: A: ¡Tengo el peor trabajo del mundo! B: Si... seguro...)

  Para resolver estos tipos de ambigüedad y otros, el problema central en el PLN es la traducción de entradas en lenguaje natural a una representación interna sin ambigüedad, como árboles de análisis o "*word embeddings*", que son vectores numéricos generados a partir de palabras.

- **Detección de separación entre las palabras:** Cuando se utiliza el lenguaje oral, es decir, cuando se habla, no se suelen hacer pausas entre palabra y palabra, por lo que la separación de las mismas dentro de una frase depende de la posibilidad de que la palabra mantenga un sentido lógico, tanto gramatical como contextual, dentro de la frase. Mientras tanto, en la lengua escrita, idiomas como el chino mandarín o el japonés no tienen separaciones entre las palabras. Esto significa que un ordenador necesita de algún tipo de ayuda para poder reconocer qué forma parte de una palabra y que forma parte de otra, lo cual puede llevar a problemas a la hora de analizar los datos teniendo en cuenta de que, en algunos lenguajes, existen las llamadas palabras compuestas, como las palabras "milk","shake" y "milkshake" en el Inglés.

[2] `http://dle.rae.es/?id=4wkKYpU`

- **Recepción imperfecta de datos:** Los acentos y ortografía, regionalismos o dificultades del habla, errores de mecanografiado o expresiones no gramaticales, errores en la lectura de textos mediante Reconocimiento Óptico de Caracteres (OCR).

- **Lenguaje no literal (lenguaje figurado):** Cuando una palabra expresa una idea en términos de otra, apelando a una semejanza que puede ser real o imaginaria. Por ejemplo, en la frase "El mar de tus ojos" la palabra "mar" se utiliza para decir que la persona tiene los ojos azules, lo cual un ordenador no puede reconocer de la misma forma que una persona.

2.1.2. Historia del PLN

Los inicios del PLN comenzaron en 1950, cuando Alan Turing publicó "Computing machinery and intelligence" [Turing, 1950], en el que se proponía el hoy conocido como "Test de Turing" que utilizamos como criterio de inteligencia. Dicho test consiste en una prueba en la que la máquina debe mostrar un comportamiento inteligente lo más parecido a un ser humano.

Turing propuso que un humano evaluara conversaciones en lenguaje natural entre un humano y una máquina diseñada para generar respuestas similares a las de un humano, limitando la respuesta al medio textual. En caso de que el evaluador, el cual sabe que uno de los interlocutores es una máquina, no pueda distinguir entre el humano y la máquina acertadamente, la máquina habría pasado la prueba.

Desde entonces y hasta la década de 1980, la mayoría de los sistemas de PLN se basaban en un complejo conjunto de reglas diseñadas a mano. Sin embargo, a partir de finales de 1980 hubo una revolución en PLN con la introducción de algoritmos de aprendizaje automático para el procesamiento del lenguaje.

Hoy en día, el incremento en el uso de redes neuronales y Machine Learning (ML) ha facilitado el desarrollo de métodos más rápidos y eficientes para tratar distintos problemas del PLN, lo que ha impulsado el desarrollo de nuevas herramientas de reconocimiento de texto como el OCR y el desarrollo de una IA más avanzada.

Entre las distintas tareas que se investigan en relación al PLN, existen algunas que se centran en la generación automática de información, como es el campo de Generación del Lenguaje Natural, o la generación de resúmenes, que comentamos a continuación.

2.2. Generación de Resúmenes

Una de las tareas que ocupa el PLN es la generación de resúmenes, que tiene como objetivo la condensación automática de información manteniendo los factores o partes más relevantes de la misma [Lloret, 2011]. Por esta razón se encarga también de investigar maneras eficientes para automatizar el proceso de síntesis por medio de ordenadores y computación.

Los humanos sintetizamos grandes cantidades de información, tanto visual como de todos nuestros sentidos, de forma natural, reconociendo las partes más relevantes de ésta

para quedarnos con lo que más nos interesa, lo que comúnmente se conoce como "separar la paja del trigo".

El principal problema que existe en este campo es que la información textual es abstracta en sí misma, de modo que no hay manera de decir qué parte de la información recibida es o no relevante. Además de esto, están también los problemas mencionados anteriormente que son generales a todo el PLN y que incluyen la problemática de la subjetividad, ya que puede que la información relevante para una persona, otra pueda verla como inútil o innecesaria.

Para intentar esclarecer este tipo de incógnitas existen ya algunas propuestas en las que se analizan algunos de los posibles aspectos de la información de entrada, evaluándola de manera que se establezca un orden de mayor a menor relevancia y mostrando únicamente la información más relevante al usuario final.

2.3. Propuestas existentes

En esta sección hablaremos sobre las propuestas que existen actualmente en cuanto a la generación de resúmenes concierne.

En la actualidad, existen multitud de sistemas de resúmenes que trabajan sobre texto plano, tanto locales, que trabajan en la máquina anfitriona, como online, que normalmente suelen ser demos que consumen servicios web.

Estos sistemas utilizan diversos algoritmos, cada uno diferente al anterior, para conseguir realizar un resumen que pueda complacer las necesidades del usuario.

A continuación listamos algunos sistemas de generación de resúmenes que existen:

- Online:

  Automatic Text Summarizer - `http://www.autosummarizer.com`

  Free Summarizer - `http://www.freesummarizer.com`

  ExplainToMe - `http://open.blockspring.com/jjangsangy/summarize-text`

  Text Summarizer Online - `www.textsummarization.net/text-summarizer`

  Compendium [Lloret, 2011] - `http://gplsi.dlsi.ua.es/demos/compendium/`

- Locales:

  SemPCA-summariser - Herramienta del GPLSI basada en el trabajo de [Alcón and Lloret, 2015]

  SummaNLP [Barrios et al., 2016] - `https://github.com/summanlp/textrank/tree/master/summa`

  libOTS [Dom Lachowicz and Rotem, 2008] - `https://github.com/neopunisher/Open-Text-Summarizer`

  MEAD – `www.summarization.com/mead`

  TextRank [Mihalcea and Tarau, 2004] – `https://github.com/davidadamojr/TextRank`

Por ejemplo, en el caso de MEAD, no es un algoritmo en sí, sino un framework público, es decir, un entorno de trabajo, escrito en Perl, que engloba un conjunto de herramientas para resumir textos variados.

En sus inicios, se desarrolló en base a un enfoque basado en el centroide, en el que se representan las frases de un texto como puntos en el espacio, de los cuales se obtiene el centroide, seleccionando las frases más cercanas a él como parte del resumen.

Sin embargo, hoy en día se han implementado muchos otros algoritmos dentro del framework MEAD (véase LexRank[3]) e incluso puedes implementar tu propio algoritmo y experimentar con él dentro del framework de MEAD.

Por otro lado, incluso al no ser herramientas locales que se puedan configurar minuciosamente como MEAD, algunas de las herramientas online existentes, como la mayoría de las arriba comentadas, permiten algún tipo de configuración como, por ejemplo, la más común de ellas, el número máximo de líneas que se desea que tenga el resumen final.

En el caso específico de la herramienta Compendium, permite seleccionar el nivel de análisis que se realiza del texto (análisis de redundancia, identificación de tópicos y detección de relevancia) y el porcentaje de compresión deseado, lo cual ayuda a configurar un tipo de resumen más adaptado a lo que el usuario puede desear.

Es debido a que existen tantísimos sistemas genéricos y al hecho de que cada uno de ellos utiliza técnicas distintas para determinar la relevancia (o no) del texto analizado, que es muy difícil compararlos y analizarlos todos.

Centrándonos ya más en el ámbito que nos concierne y como la literatura sobre esto es ya bastante amplia, he decidido centrarme únicamente en la generación de resúmenes multiperspectivos (que ofrecen opiniones positivas y negativas) y del ámbito hotelero.

Esta decisión se debe al hecho de que, entre todos los tipos de resúmenes existentes, los resúmenes multiperspectiva son los más recientes en lo relacionado al UGC ya que permiten la observación de la diversidad de opinión existente entre varios usuarios generadores de contenido con respecto a un tema específico.

Por ejemplo, [Lloret, 2016] propone un método inicial para la explotación de los metadatos existentes en la información recolectada y su uso en la generación de resúmenes multiperspectiva. Esto es, la generación de resúmenes enfocados de distinta manera dependiendo de las necesidades del usuario en concreto.

Para ello realizó un estudio en el que terminó subdividiendo el proceso de generación de resúmenes en 3 fases:

- Procesamiento de texto
- Análisis del contenido semántico
- Generación del resumen

En la primera de estas fases se propone la extracción de información básica de los textos, como es la detección del tópico de una frase o fragmento de la misma, la detección

[3] `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume22/erkan04a-html/erkan04a.html`

de la polaridad de la frase y la identificación de lo que ella llama Basic Information Units (BIUs) o unidades básicas de información.

Estas BIUs están formadas por fragmentos de texto, que pueden ir desde frases completas a clausulas, tripletas Sujeto-Verbo-Objeto (SVO) o palabras clave (keywords). A partir de ellas se realiza un proceso de detección de tópico y otro de detección de la polaridad de la BIU.

Haciendo uso de esta información, se realiza una identificación de BIUs comunes y contradictorias basándose en un conjunto de heurísticas y centrándose en aquellas que expresan una opinión subjetiva.

Este proceso es necesario para el fin de seleccionar las BIUs más relevantes que se utilizarán en la generación del resumen final a partir de una plantilla.

En el trabajo presentado por [Esteban and Lloret, 2017a] también se propone un sistema similar al de este TFG llamado TravelSum[4], que consiste en una aplicación web a través de la cual los usuarios pueden obtener un resumen generado automáticamente en Español teniendo en cuenta la información proporcionada por otros usuarios [Esteban and Lloret, 2017b].

Utilizando como fuente de datos la red social Twitter[5] y TripAdvisor[6], su sistema comienza segmentando las frases con la herramienta Stanford CoreNLP[7] [Manning et al., 2014]. Tras eso, realizan un análisis y clasificación de polaridad para cada frase utilizando la herramienta propuesta por [Fernández et al., 2015].

Tras obtener la polaridad de cada frase, se agrupan según esta y, a partir de ellos se calcula la similitud entre frases usando el método del coseno que proporciona la librería Simmetrics [8].

Además de lo arriba mencionado, este sistema aplica una serie de fórmulas de relevancia, según el tipo de fuente a la que pertenece la información y la relevancia de la frase respecto a su propia fuente de información.

Además se calcula la intensidad de la frase, que obtienen del análisis de la polaridad anteriormente mencionado y la aparición de complementos del sustantivo, para realizar la selección de las frases que se procesarán para volverlas más impersonales. Dicho proceso consiste en una serie de reglas en las que se convierte la frase a tercera persona del plural a partir de la extracción de información morfológica presente en la propia frase [Esteban and Lloret, 2017a].

Finalmente, utilizando frases predefinidas se crean 3 resúmenes dependiendo de lo que el usuario desee conocer: los aspectos positivos, los aspectos negativos o un resumen mixto con ambos.

Este sistema se parece bastante al que se propone en este TFG al hacer análisis de polaridad y tópico. Sin embargo, existen puntos en los que se diferencian. Por ejemplo, TravelSum es un sistema de generación de resúmenes multigénero, mientras que el propuesto en este TFG es un sistema de generación de resúmenes multiperspectiva. Esto

[4] `http://travelsum.gplsi.es/`
[5] `https://twitter.com/`
[6] `https://www.tripadvisor.es/`
[7] `https://stanfordnlp.github.io/CoreNLP/`
[8] `https://github.com/Simmetrics/simmetrics`

es, que TravelSum se centra en el análisis de distintas fuentes de información para la generación del resumen, mientras que el sistema propuesto se centra en la diferencia de opiniones existentes dentro de una misma fuente de información.

Por otra parte, TravelSum únicamente agrupa frases por su polaridad, mientras que nuestra propuesta va más allá, teniendo en cuenta las diferentes opiniones de los usuarios así como el tipo de usuario que realiza un comentario, lo cual comentaremos en capítulos siguientes.

Por último, [Hu et al., 2017] propone un sistema consistente en 4 fases. En la primera de ellas preprocesa su corpus, obtenido de TripAdvisor, para eliminar las stopwords[9] y obtener, a partir de eso, los nombres, adjetivos y adverbios negativos a través de etiquetado gramatical, conocido más comúnmente como Part of Speech (PoS) Tagging, que consiste en el proceso de asignar a cada una de las palabras de un texto su categoría gramatical[10].

Tras ello, en la segunda fase del proceso, realiza una selección de frases basándose en una serie de reglas que calculan la importancia de cada frase teniendo en cuenta la reputación del autor, la utilidad de la crítica, cuando se publicó dicha crítica y su contenido.

Para saber si dos frases están hablando sobre lo mismo, se realiza un análisis de la similitud de las frases en la tercera fase de su proceso, en la que tiene en cuenta dos tipos de similitud, en contenido y en sentimiento. La similitud en contenido la obtiene utilizando Normalized Google Distance (NGD), obteniendo un conjunto de palabras clave utilizadas para definir la disparidad entre dos términos, mientras que la similitud de sentimiento es calculada mediante Semantic Orientation Pointwise Mutual Information (SOPMI) para calcular la puntuación de sentimiento positivo y negativo para cada frase.

Por último se seleccionan las frases particionándolas en k grupos relacionados (clústers) usando el algoritmo k-medoid[11], calculando la puntuación de relevancia, sumando la de cada frase y seleccionando las top-k frases más representativas de los top-k clústeres.

2.4. Limitaciones de las propuestas existentes

Algunas de las principales limitaciones de las propuestas existentes yacen en el hecho de que, al tratarse del lenguaje escrito, no es sencillo saber cuándo una frase o tópico es contradictorio debido a que el lenguaje en sí mismo es ambiguo.

Eliminar por completo esta ambigüedad en el lenguaje es imposible por el momento, pero en este TFG proponemos un sistema que analiza la contradicción entre frases para que así sea posible discernir entre la información que es útil y la que no lo es teniendo en cuenta el perfil del usuario.

El sistema propuesto por [Lloret, 2016] es una buena base para partir en lo que a generación de resúmenes respecta, sin embargo, [Esteban and Lloret, 2017a] deja algunos

[9] Las stopwords son palabras que, al no tener significado por sí solas, no proporcionan ningún dato importante al análisis. Dichas stopwords suelen ser preposiciones, pronombres y artículos entre otras.

[10] `https://es.wikipedia.org/wiki/Etiquetado_gramatical`

[11] `https://es.wikipedia.org/wiki/K-medoids`

puntos que podrían ser mejorados.

[Esteban and Lloret, 2017a] comenta que su principal problema fueron los errores ortográficos y gramaticales, los cuales aparecían de vez en cuando cuando su sistema intentaba transformar la frase a tercera persona. Estos errores suelen interferir a la hora de realizar una correcta detección del tipo de palabra, ya que el diccionario que utilizan estos sistemas puede fallar si la palabra no está escrita correctamente.

Por otra parte, [Hu et al., 2017] hace uso de un sistema de reputación, sin embargo también tiene el problema de no realizar un análisis de tópicos, utilizando un algoritmo de similitud en su lugar.

En la propuesta de este TFG se hace uso de un corrector ortográfico que elimina, en la medida de lo posible, los errores ortográficos existentes en los textos generados por los usuarios.

De este modo aseguramos un mayor índice de acierto a la hora de detectar el tipo de cada palabra existente en una frase, así como el tópico principal de la misma.

3. Objetivos

3.1. Generales

El trabajo que se propone en este TFG consiste en analizar técnicas de procesamiento del lenguaje natural para construir un método de generación de resúmenes automático que permita ofrecer los distintos puntos de vista sobre opiniones realizadas por usuarios (por ejemplo, hoteles, películas, restaurantes, etc.) previamente, analizando la variedad de opiniones existentes.

3.2. Específicos

Los objetivos del proyecto son:

- Aprender nociones básicas sobre el procesamiento del lenguaje natural y de las tecnologías del lenguaje humano, orientadas a la clasificación de información y generación de resúmenes.
- Adquirir conocimientos relacionados con la recuperación y extracción de información.
- Analizar distintas fuentes de información y seleccionar una (o varias) para la creación de un conjunto de datos sobre el que trabajar.
- Analizar y decidir las tecnologías más apropiadas para desarrollar el trabajo.
- Aprender y utilizar herramientas y librerías específicas de Procesamiento de Lenguaje Natural.
- Desarrollar un método novedoso de generación de resúmenes automático que genere resúmenes más adecuados que los sistemas que actualmente existen.
- Evaluar el método propuesto para determinar la calidad y adecuación de los resúmenes generados.

4. Metodología

En este capítulo explicaremos la metodología que se ha utilizado para el desarrollo de este TFG y cuáles han sido las tecnologías utilizadas para ello.

4.1. Metodología de desarrollo

Durante todo el desarrollo de este TFG se ha seguido una metodología ágil basada en Scrum[1].

Las metodologías ágiles se enfocan en la entrega de pequeños trozos de trabajo cada poco tiempo, en iteraciones, ordenando las tareas existentes por prioridad y realizando las que corre más prisa tener listas.

Semanalmente se realizaron reuniones con la profesora Elena Lloret para discutir los distintos aspectos del trabajo, la propuesta del método y el estado del sistema.

En cada reunión se identificaba qué se había realizado, qué faltaba por hacer y qué parte se iba a abordar hasta la próxima reunión.

Algunos de los aspectos que se abordaron en las reuniones fueron las herramientas que se podrían utilizar y qué lenguajes de programación serían viables para el desarrollo de este trabajo.

Finalmente se llegó a un consenso y en la siguiente sección veremos cuáles de estos fueron utilizados finalmente.

4.2. Tecnologías utilizadas

En esta sección hablaremos de las distintas herramientas y lenguajes de programación que han sido utilizados para el desarrollo de este TFG.

En la siguiente tabla se encuentran enumeradas las tecnologías existentes que hemos utilizado en nuestro sistema y en qué etapa del mismo son utilizadas.

Asimismo, también existen algunas herramientas que terminaron siendo descartadas en el proceso de selección y que se mencionan en otra tabla más adelante.

Como se menciona en la Tabla 4.1, se han utilizado dos lenguajes de programación para el desarrollo de este TFG, Python y Java, mencionando también las etapas del proceso de generación, explicado en la sección 5.1 de este TFG, en la que se utilizan.

La mayor parte del trabajo ha sido desarrollada en Python, sin embargo, en el apartado de corrección ortográfica se decidió usar Java, ya que nos daba la posibilidad de

[1] `https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)`

reutilizar código ya desarrollado que permitiese la corrección de las palabras de una manera automática. Este proceso se explicará más en detalle en futuras secciones.

Nombre	Descripción		Enlace/URL	Etapa
Java 8	Lenguaje de programación orientado a objetos		`https://www.java.com/es/`	Corrección Ortográfica
Python 2.7	Lenguaje de programación levemente interpretado con tipado dinámico		`https://www.python.org/`	Análisis Lingüístico Interpretación Generación del Resumen
MySQL	Sistema de gestión de Base de Datos (BBDD)		`https://www.mysql.com/`	Recolección del corpus
SENSEI	Crawler (araña web) [2] desarrollado por el Grupo de Procesamiento del Lenguaje natural y Sistemas de Información (GPLSI)		`https://gplsi.dlsi.ua.es/gplsi13/es/node/301`	Recolección del corpus
NLTK	Plataforma para construir programas escritos en Python que trabajen con datos procedentes del lenguaje humano.	- WordNet - SentiWordNet - WordNetLemmatizer - Tokenizer	[Bird and Loper, 2004] `https://www.nltk.org/`	Análisis Lingüístico
Gensim	Librería python con un amplio conjunto de herramientas útiles para la detección de tópicos en un texto		[Řehůřek and Sojka, 2010] `https://radimrehurek.com/gensim/`	Análisis Linguístico
GloVe	Algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. Dispone de varios conjuntos pre-entrenados útiles para la de vectorización de palabras (Word2Vec)		[Pennington et al., 2014] `https://nlp.stanford.edu/projects/glove/`	Análisis Linguístico

Tabla 4.1.: Tabla de Tecnologías Utilizadas

Esta tabla ha sido elaborada teniendo en cuenta varios aspectos, como que Python es el lenguaje de programación más versátil en lo que a PLN se refiere, o que la plataforma Natural Language Toolkit (NLTK) agrupa la mayoría de las herramientas necesarias sin tener que depender de multitud de herramientas de difícil instalación.

Cabe aclarar que, para el almacenamiento de los datos y tener un acceso más rápido

[2] `https://es.wikipedia.org/wiki/Ara%C3%B1a_web`

a ellos, se hizo uso de una base de datos MySQL instalada en la máquina local y que fue creada con un script que se encuentra en el apartado A.6 del anexo con el diseño indicado en la Figura 4.1 que se muestra a continuación.

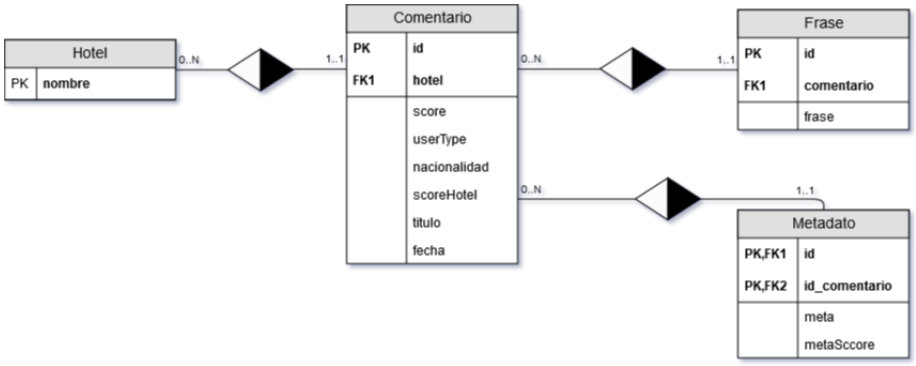

Figura 4.1.: Esquema de Base de Datos

Este esquema utiliza una notación que combina los Esquemas Entidad-Relación (EER)[3] con esquemas UML[4]. Esta combinación permite una visualización más ordenada del esquema de datos a la vez que permite diferenciar las relaciones entre todas las tablas.

Se debe tener en cuenta de que el script depende de que la herramienta MySQL esté correctamente instalada y en la ruta indicada por el mismo. En caso de que esté instalada, pero no esté en la ruta que especifica el script, deberá modificarse esta ruta antes de ejecutar el mismo.

Por otra parte, hubieron tecnologías, mencionadas en la Tabla 4.2, que fueron analizadas para probar si se podían integrar en el sistema propuesto, pero que finalmente se descartaron por su baja curva de aprendizaje, es decir, por la dificultad de aprender cómo se utilizaban, o porque otras herramientas realizaban el mismo trabajo con menor coste temporal.

Nombre	**Descripción**	**Enlace/URL**
Keras	API de redes neuronales a alto nivel, escrita en Python y capaz de funcionar sobre TensorFlow, CNTK, o Theano. Fue desarrollada con el objetivo de permitir una experimentación rápida.	[Chollet et al., 2015] `https://keras.io/`
Scikit-Learn (SKLearn)	Conjunto de herramientas de código abierto para minería y análisis de datos construida sobre NumPy, SciPy y matplotlib	[Pedregosa et al., 2011] `http://scikit-learn.org/stable/index.html`

Tabla 4.2.: Tabla de Tecnologías Descartadas

[3] `https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n`
[4] `https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado`

Nombre	Descripción	Enlace/URL
FreeLing	Librería en C++ que provee funcionalidades para el análisis del lenguaje (análisis morfológico, detección de entidades nombradas, etiquetado PoS, parseo, desambiguación del sentido de la palabra, Semantic Role Labelling (SRL), etc.) para una gran variedad de lenguajes entre los que se encuentran el Inglés y el Español entre muchos otros.	[Padró and Stanilovsky, 2012] `http://nlp.lsi.upc.edu/freeling/`

Tabla 4.2.: Tabla de Tecnologías Descartadas

En un principio se tuvo la intención de utilizar redes neuronales para la detección de la contradicción entre dos frases, pero debido a la complejidad del problema se decidió dejarlo para un futuro próximo.

En una de las fases que comentaremos a continuación se realizó un clúster de términos, es decir, un pequeño programa que se utilizaba para agrupar los términos relacionados. Para poder encontrar un clúster que nos fuera útil, se realizaron varias pruebas, entre las cuales se hacía uso de las herramientas de Scikit-Learn, mencionadas en la anterior tabla. Finalmente se decidió descartar Scikit-Learn porque no realizaba correctamente la tarea del clústering.

Por otra parte y ya que la librería NLTK nos proporcionaba ya un PoS Tagger se decidió prescindir de Freeling.

Todo este análisis y error nos sirvió para perfeccionar la propuesta de nuestro sistema de generación de resúmenes objetivos, la cual explicamos en la siguiente sección.

En el siguiente capítulo daremos paso a una explicación más detalla sobre qué consiste este TFG.

5. Cuerpo del trabajo

En este capítulo explicaremos más en profundidad el enfoque que se ha seguido para la propuesta del sistema generador de resúmenes objetivos.

5.1. Propuesta para la generación de resúmenes objetivos

En esta sección explicaremos la propuesta que hemos desarrollado en este TFG para diseñar y desarrollar un sistema de generación de resúmenes objetivos.

Según [Hovy, 2003], el proceso de generación de resúmenes se divide en 3 etapas fundamentales:

- Identificación del tópico
- Interpretación
- Generación del resumen

Cada una de estas etapas separa los diferentes aspectos necesarios para la generación de un resumen de la manera en la que lo haría un ser humano.

La etapa de "Identificación del tópico" analiza el texto para buscar la información relevante para el usuario. La etapa de "Interpretación" toma esta información y la organiza para que, en la etapa de "Generación del resumen", se tome únicamente la más relevante a la hora de crear un nuevo resumen.

Debido a que el sistema propuesto en este TFG para la generación de resúmenes objetivos está pensado desde el punto de vista del usuario final, se decidió tomar las fases propuestas por [Hovy, 2003] como base para dicha propuesta.

Como punto diferenciador con el resto de sistemas generadores de resúmenes, se pensó en la posibilidad de crear resúmenes enfocados, no solamente en el público genérico, sino en un conjunto específico de personas, para poder dar un resumen más adaptado a las necesidades del usuario.

Por esto, se analizó el método que utilizaría un humano para obtener la información relevante de un conjunto de datos y se propuso el siguiente sistema:

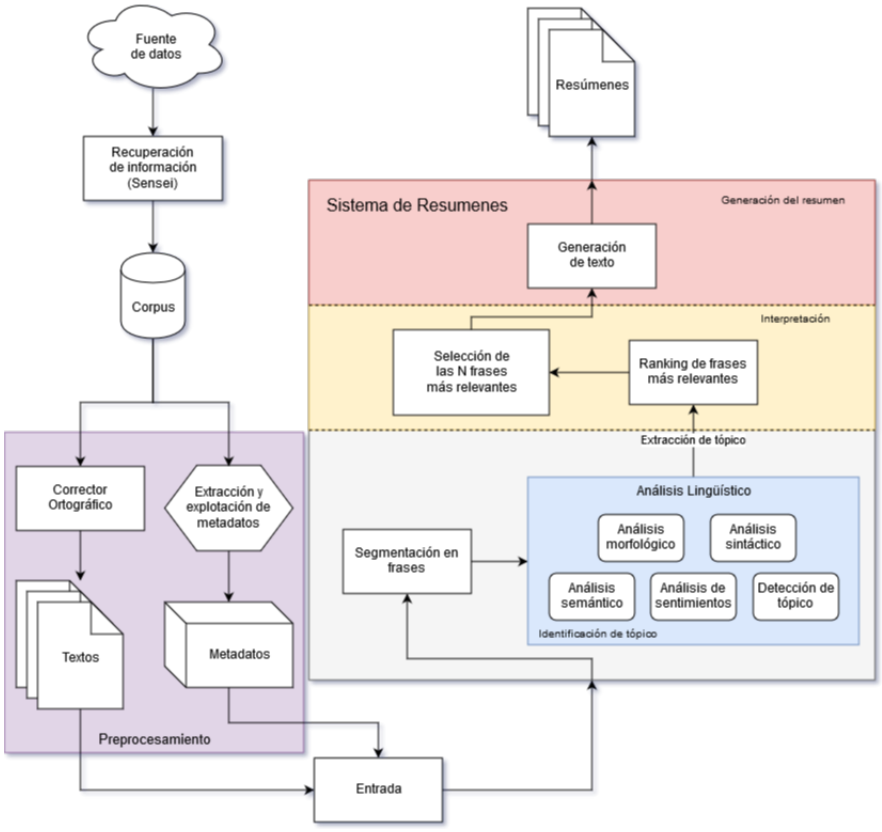

Figura 5.1.: Esquema del sistema propuesto

Primero, preprocesamos la información para tener un conjunto de datos lo más limpio posible antes de introducir estos datos a nuestro sistema.

Una vez dentro de nuestro sistema la información es segmentada para su posterior análisis, el cuál utilizamos como base para obtener un ranking de las frases más relevantes.

De este ranking seleccionaremos un conjunto de las más relevantes entre todas para generar un resumen adaptado al usuario a partir de plantillas.

En las siguientes secciones explicaremos más en detalle cada parte del sistema y las diferentes tecnologías que se han utilizado.

5.1.1. Corpus

Para poder realizar cualquier tipo de análisis es necesario un conjunto de datos que sirvan como base para realizarlo. En PLN se utiliza el nombre “Corpus” para referirnos a este conjunto de datos.

En esta sección comentaremos cómo ha sido la selección y recolección de dicho corpus utilizado para el desarrollo de este TFG.

5.1.1.1. Recolección del corpus

El corpus en el cual se basa este TFG fue recolectado automáticamente utilizando el crawler SENSEI[1], desarrollado por el GPLSI[2], en su versión 3.0.

Con la ayuda de este crawler hemos recolectado desde la página británica de TripAdvisor[3] la totalidad del corpus, el cual, para poder realizar un estudio más controlado, se decidió recolectar acotando una zona geográfica específica.

Cabe mencionar que se decidió que nuestro corpus estuviera en Inglés debido a que es uno de los idiomas que más se hablan en todo el mundo, de modo que nos proporcionaría una mayor cantidad de información si recopilábamos el corpus en este idioma. Además, el hecho de que la mayoría de las herramientas de PLN existentes estén en Inglés fue también un factor importante a la hora de tomar esta decisión.

Como zona geográfica se escogió Roma, Lazio por la densidad de población y visitantes de la zona, ya que nos permitiría obtener una mayor cantidad de datos que otros lugares, mientras manteníamos el corpus en una densidad no demasiado grande para su uso inicial.

En nuestro caso, el corpus consta de los comentarios hechos por los usuarios de los 10 hoteles especificados en la Tabla 5.1, los cuales hemos recolectado teniendo en cuenta si el usuario pertenece a alguno de los siguientes grupos demográficos:

Familias Parejas Solitario Negocios Amigos

De este modo nos es más sencillo acotar el tipo de usuario que comenta para así realizar una aproximación dependiente del grupo en cuestión.

Hotel San Remo	Black hotel
Hotel Columbus	Comfort Hotel Bolivar
Excel Hotel Roma Montamario	Grand Hotel Fleming
Sina Bernini Bristol	Hotel Turner
Quality Hotel Nova Domus	Hotel Lord Byron

Tabla 5.1.: Hoteles del corpus

A continuación se muestra en la Figura 5.2 los datos recolectados de cada uno de los comentarios.

[1] `https://gplsi.dlsi.ua.es/gplsi13/es/node/301`
[2] `https://gplsi.dlsi.ua.es/gplsi13/es`
[3] `http://www.tripadvisor.co.uk`

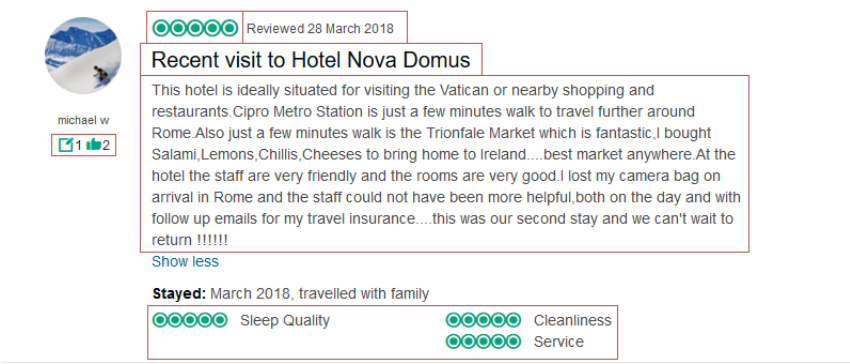

Figura 5.2.: Datos recolectados de un comentario

En dicha figura se muestran los siguientes datos:

- Metadatos
  - Fecha de publicación
  - Número de contribuciones *
  - Votos sobre la utilidad del comentario *
  - Valoración general sobre el hotel
  - Valoración sobre distintos aspectos del hotel *
- Información textual
  - Título del comentario
  - Comentario

Además de los datos anteriormente mencionados, también almacenamos la fecha en la que fueron recolectados los datos, útiles en los casos en los que la fecha de publicación indica "Days ago", el nombre del hotel y el tipo de usuario al que pertenece el comentario. En ocasiones también se muestra la procedencia del usuario, que también es recolectada en caso de existir.

Como los datos que se recolectaron se guardaron posteriormente en un fichero de texto plano, se utilizó una estructura sencilla para poder diferenciar entre las distintas partes de un comentario y comentarios distintos.

Dicha estructura siguió el formato visible en el Listado 5.1, teniendo en cuenta que cada línea supondría una línea del fichero final.

*En algunos casos este valor no existe, por lo que no se recolecta

Listado 5.1: Estructura de recolección de corpus

```
1 Título del comentario | Tipo de usuario | Valoración general sobre
      el hotel | fecha de publicación | procedencia del usuario | nú
    mero de contribuciones | votos sobre la utilidad del comentario
2 Comentario
3 @metadato1 puntuación1 @metadato2 puntuación2 @metadato3....
```

En la primera línea se almacenaron la mayoría de los datos importantes del comentario, mencionados anteriormente. En la segunda línea se guardaría la totalidad del comentario, uniendo todas las líneas del mismo en una. Por último, la tercera línea guardaría los metadatos de las valoraciones sobre los distintos hoteles.

Para poder distinguir entre las distintas partes existentes en cada línea, se hizo uso de caracteres especiales como son " |" (Pipe), " @ " (Arroba) y los espacios y saltos de línea.

Además, para diferenciar los datos entre comentarios, éstos se separaban con una línea en blanco entre cada uno de ellos.

Sabiendo esto, en la Tabla 5.2 mostramos la cantidad de comentarios obtenidos para cada hotel y cada tipo de usuario.

Debemos aclarar que estas estadísticas son referentes a nuestro corpus, recolectado el día 23 de Noviembre de 2017, por lo que es muy probable que estas estadísticas no sean exactas a día de hoy debido a que esta información se actualiza constantemente.

Hotel	**Familias**	**Pareja**	**Solitario**	**Negocios**	**Amigos**	**Total**
Black hotel	52	131	10	19	42	254
Comfort Hotel Bolivar	111	163	20	8	47	349
Excel Hotel Roma Montemario	79	146	16	18	46	305
Grand Hotel Fleming	23	60	10	7	33	133
Hotel Columbus	118	144	20	26	61	369
Hotel Lord Byron	73	342	20	30	32	497
Hotel San Remo	86	205	37	7	65	400
Hotel Turner	81	322	21	20	63	507
Quality Hotel Nova Domus	76	130	11	10	72	299
Sina Bernini Bristol	133	236	21	50	62	502

Tabla 5.2.: Estadísticas del corpus (23/11/2017)

Podemos observar que la cantidad de usuarios que han realizado comentarios son más abundantes para los tipos de usuario "Familias" y "Pareja". Esto se debe a que la ciudad

en cuestión es un punto turístico que suelen visitar parejas y familias más que gente en solitario o en visitas de negocios.

Una vez ya recolectado el corpus, fue preprocesado antes de poder introducirlo dentro de nuestro sistema, como comentamos en la siguiente sección.

5.1.2. Preprocesado de datos

En esta sección comentaremos el preprocesado que se llevó a cabo en el corpus para corregir los errores ortográficos y procesar los metadatos que se recolectaron antes de poder introducir dichos datos en nuestro sistema.

5.1.2.1. Corrección ortográfica

Para evitar en la medida de lo posible los errores ortográficos causados a la hora de escribir los comentarios, se ha pasado el corpus por un corrector ortográfico basado en la librería Jazzy[4] para Java. De este modo nos podemos asegurar que los datos recopilados tienen el menor número de errores posibles antes de comenzar el procesado de los mismos.

Este corrector depende de un fichero de texto que utiliza a modo de diccionario para poder corregir los posibles errores. En nuestro caso hemos usado un diccionario en Inglés[5] ligeramente modificado para añadir algunas siglas, como puede ser "WiFi", y palabras que son específicas del ámbito y que no queremos corregir.

Al ser un corrector automático, puede darse el caso de que la corrección elegida no sea exacta.

Por ejemplo, la frase "*very nice and the decor is truly different.A bit far from the nearest metro station*" es corregida por "*very nice and the decor is truly different bit far from the nearest metro station*".

En este caso la palabra "*different*", el punto final de la frase y el inicio de la siguiente frase, no están debidamente separados, de modo que el corrector lo detecta como un error de escritura y elimina la primera palabra de la siguiente frase.

El código de esta subsección se puede encontrar en los apartados A.1 y A.2 del anexo para su consulta.

Una vez finalizada la corrección de la información recolectada, podemos pasar al procesado de los metadatos y la introducción de esta información en el sistema de generación de resúmenes.

5.1.2.2. Procesado de metadatos

Tras haber limpiado el conjunto de datos, se procede a realizar un preprocesado de los mismos.

Como hemos comentado anteriormente en la sección 5.1.1.1, el corpus fue recolectado en formato de texto plano y se utilizaron caracteres especiales para diferenciarlos entre sí.

[4] `http://repo1.maven.org/maven2/net/sf/jazzy/jazzy-core/0.5.2/jazzy-core-0.5.2.jar`
[5] `https://introcs.cs.princeton.edu/java/data/words.utf-8.txt`

Haciendo uso de estos caracteres especiales se realizó un proceso de selección que leía la información de los comentarios y la guardaba en la base de datos, cuya estructura se menciona en la sección 4.2.

Por una parte, realizamos un proceso de tokenización del comentario en sí, es decir, separamos las distintas frases que conformaban un comentario en palabras, haciendo uso de la herramienta NLTK.

Por otra parte se hizo un procesado de los siguientes metadatos:

- Fecha de publicación
- Número de contribuciones
- Votos sobre la utilidad del comentario
- Valoración general sobre el hotel

Antes de introducir todos los datos en la BBDD hacemos el cálculo de la relevancia del autor.

Dicha relevancia se calcula desde 0 y añade más o menos puntos en función a diferentes tablas de heurística basadas en los metadatos y que mostramos a continuación:

Días (x) desde la fecha de publicación	**Puntos**
$x > 365$	1
$180 < x < 365$	2
$90 < x < 180$	3
$30 < x < 90$	4
$14 < x < 30$	5
$7 < x < 14$	6
$x < 7$	7

Tabla 5.3.: Heurística para la fecha de publicación

Número de contribuciones (x)	**Puntos**
$0 < x < 11$	1
$11 < x < 50$	2
$50 < x < 100$	3
$x > 100$	4

Tabla 5.4.: Heurística para el número de contribuciones

Porcentajes (x)	**Puntos**
$0 < x < 10$	0
$10 < x < 30$	1
$30 < x < 60$	2
$60 < x < 100$	3
$x = 100$	4
$x > 100$	5

Tabla 5.5.: Heurística para los votos sobre la utilidad del comentario

Los "Días desde la fecha de publicación" mencionados en la Tabla 5.3 se cuentan a partir del día en el que se obtuvieron los datos (23/11/2017), mientras que los porcentajes mencionados en la Tabla 5.5 se calculan con la siguiente formula:

$$\frac{Contribuciones\ útiles * 100}{Total\ de\ contribuciones}$$

Entendemos como contribuciones útiles aquellas que los demás usuarios de TripAdvisor han votado como relevantes a la hora de elegir un hotel.

La suma de todos los aspectos de las tablas anteriores nos dará la puntuación de relevancia para el autor en este comentario, como muestra la ecuación 5.1. Esto nos será útil en la selección de frases que explicaremos en una de las secciones a continuación.

$$\begin{matrix} Puntuación\ de \\ relevancia\ del\ autor \end{matrix} = \begin{matrix} Heurística\ Tabla\ 5.3\ +\ Heurística\ Tabla\ 5.4\ + \\ Heurística\ Tabla\ 5.5 \end{matrix} \tag{5.1}$$

Una vez calculada esta relevancia, se almacenará en la base de datos junto a las frases del comentario, las valoraciones individuales de los aspectos del hotel y el resto de datos recolectados.

A continuación se pasará a analizar los datos del comentario dentro del sistema propuesto, empezando por la identificación del tópico.

5.1.3. Identificación del Tópico

En esta sección veremos los distintos análisis que hemos llevado a cabo para la detección e identificación de tópicos, así como otra información relevante para la generación de resúmenes en el sistema propuesto.

5.1.3.1. Análisis morfológico, sintáctico y semántico

El análisis morfológico y sintáctico es una parte importante para la detección de información relevante relacionada con las palabras.

El saber qué tipo de palabra y qué relación tiene con el resto de las palabras de una frase o texto, puede ayudar en gran medida a detectar su relevancia dentro de un conjunto de información.

También, haciendo uso del conjunto de sinónimos y relaciones semánticas existentes entre la palabra analizada y otras, comúnmente conocidos en PLN como *synsets*, podemos detectar, mediante el análisis semántico de las palabras, la polaridad relacionada con dicha palabra, como se explicará en la siguiente sección.

Todas estas tareas son realizadas en nuestro sistema por el conjunto de librerías NLTK, de modo que podamos detectar qué tipo de palabras se están analizando y si son relevantes o no para nuestro futuro resumen.

A continuación explicamos cómo utilizamos la información recabada en esta sección para el análisis de los sentimientos.

5.1.3.2. Análisis de sentimientos

Analizar el sentimiento inherente de una frase es un paso necesario a la hora de detectar si una frase está expresando una opinión positiva o negativa. Analizando este sentimiento podemos saber si dos frases que hablen sobre el mismo tema (en nuestro caso, nos referiremos al tema como tópico), tienen opiniones iguales o contradictorias.

Por esta razón y haciendo uso de la herramienta NLTK mencionada en la Tabla 4.1 obtenemos la polaridad inherente en cada una de las frases.

Para ello primero obtenemos las frases anteriormente almacenadas en la base de datos y eliminamos los posibles elementos que podrían causar error en la herramienta, tales como etiquetas HTML y caracteres no existentes en la codificación UTF-8[6].

Una vez limpio el texto, lo *tokenizamos*, separando el texto en una lista de palabras individuales que serán pasadas por un PoS Tagger, el cual nos dirá si dicha palabra se trata de un adjetivo, un sustantivo, un adverbio o un verbo.

Una vez obtenido esto, lematizamos las palabras teniendo en cuenta el tipo de palabra que son, tomando únicamente los sustantivos, adjetivos y adverbios para ello. Este proceso consiste en, dada una forma flexionada (es decir, en plural, en femenino, conjugada, etc), hallar el lema[7] correspondiente.

El hecho de que solamente se utilicen estos tipos de palabras es debido a que los verbos, que expresan acciones, dependen del contexto para la expresión de un sentimiento y podrían hacer que la detección de la polaridad mostrara errores.

Tras esto, obtendremos los *synsets* de dicho lema haciendo uso de la librería WordNet existente en NLTK.

WordNet [Miller, 1998] es una enorme base de datos léxica en Inglés en la que sustantivos, verbos, adjetivos y adverbios son agrupados en conjuntos de sinónimos cognitivos (*synsets*), cada uno expresando un concepto distinto.

Gracias a la información que nos proporcionan los *synsets* y a la herramienta SentiWordnet [Esuli and Sebastiani, 2007] podemos obtener un número que indica la cantidad de polaridad positiva o negativa que posee cada lema. Esto nos permite saber la polaridad general de la frase al sumar todos los números de las palabras que la componen.

El código desarrollado para esta sección se puede encontrar en el apartado A.3 del anexo.

Una vez analizado la polaridad de una frase, realizaremos la detección del tópico de la frase, problema que abordamos en la siguiente sección.

5.1.3.3. Detección de tópico

La detección del tópico de una frase es el proceso por el cual se obtiene el tema principal de la clase, es decir, de qué está hablando la frase.

Debido a que las frases están escritas por humanos y a que siempre estamos pensando en varias cosas al mismo tiempo, las frases tienden a hablar de varias cosas al mismo tiempo.

[6] `https://es.wikipedia.org/wiki/UTF-8`

[7] `https://es.wikipedia.org/wiki/Lema_(ling%C3%BC%C3%ADstica)`

Nuestro sistema tenía en cuenta este problema y se decidió la utilización del tópico predominante existente en dicha frase.

Por esta razón y para poder hacer una detección de tópicos relevantes, lo primero que hacemos es eliminar las partes de la frase que obstaculizarían el correcto análisis de los datos, tales como *stopwords* y signos de puntuación.

Para ello, utilizamos el corpus de *stopwords* (palabras sin significado propio como artículos, pronombres y preposiciones) proporcionado por la librería NLTK para el idioma Inglés, ya que nuestro corpus está en este idioma, y eliminamos también los signos de puntuación.

Una vez realizada esta limpieza, obtendremos los lemas de cada una de las palabras haciendo uso del WordNetLemmatizer que nos proporciona también uno de los módulos de la librería NLTK.

A diferencia del análisis de sentimientos, aquí se lematizan todas las palabras que no sean *stopwords* para el análisis de los tópicos.

Esta lista de lemas conformarán el diccionario del documento que se pasará a un modelo Latent Dirichlet Allocation (LDA)[8], el cual nos proporciona otra librería de Python llamada gensim, mencionada también en la Tabla 4.1, y que nos devolverá una lista de los tópicos más frecuentes en la frase.

Tras haber obtenido la lista de tópicos relevantes en la frase, utilizamos el PoSTagger de NLTK para saber si alguna de las palabras seleccionadas NO es un sustantivo, en cuyo caso se descartará del conjunto. Esto se debe a que los verbos, adjetivos y adverbios no se pueden utilizar para la definición de una característica, ya que, o expresan acciones o se utilizan para describir o modificar un sustantivo o verbo.

Sin embargo, estos tipos de palabras no se descartaron anteriormente porque la lista de lemas que se introducen en el LDA no debe ser únicamente de sustantivos. Esto es debido a que el modelo mira las relaciones existentes entre las palabras para su clasificación. En el caso de que solamente fueran sustantivos, no se podría analizar correctamente la relación entre las palabras y la detección de los tópicos saldría errónea.

Como el conjunto de palabras que nos devuelve gensim está ordenado por relevancia, calculada a partir del número de apariciones de la palabra en específico, utilizaremos la primera palabra del conjunto como el tópico de la frase.

En algunos casos, gensim no detecta ningún término que cumpla los requisitos de lo que nosotros entendemos como tópico en este TFG, de modo que dicho tópico para esta frase quedará vacío.

Esto se debe a que gensim ha detectado únicamente palabras que no son sustantivos como tópicos principales de la frase, de modo que quedan descartadas y el tópico para esa frase se deja vacío.

Por ejemplo, en la frase "*They really seemed not to be amazed*" se detectan los tópicos "*amaze*" y "*seem*" los cuales, al ser verbos, quedan descartados y el tópico de la frase queda como vacío.

Por último, el tópico seleccionado se introduce en la base de datos para su posterior consulta, de modo que se agilice el proceso de generación de resúmenes posterior.

[8] `https://es.wikipedia.org/wiki/Latent_Dirichlet_Allocation`

Todo el código mencionado en esta sección se puede encontrar en los apartados A.4 y A.5 del anexo.

5.1.3.4. Análisis de contradicción

Además de la detección del tópico, se realizó un análisis de la contradicción que analizaba las frases dados un hotel y un tipo de usuario en específico.

Gracias a estos datos nos podríamos asegurar si existía contradicción entre usuarios de un mismo hotel que estuvieran en las mismas condiciones de viaje.

Para realizar este análisis, utilizamos los datos recolectados anteriormente, es decir, tanto el tópico principal de la frase como el sentimiento inherente de la misma, para clasificar las frases de un mismo tópico en frases con sentimiento positivo o negativo. Esto quiere decir que, primero, agrupamos las frases por tópico, y después dividimos estos grupos en frases con polaridad positiva y frases con polaridad negativa.

Por tanto, se consideró que existía contradicción si, para un tópico dado, había al menos una frase con sentimiento positivo y otra con sentimiento negativo.

A continuación se muestran algunos ejemplos de frases que se consideran contradictorias con respecto a su tópico para un mismo tipo de usuario en un hotel específico:

Para el tópico "**restaurant**":

- Positivo: "The location is great, a lot of restaurants near and breakfast is included."
- Negativo: "Great location for sightseeing however not a huge choice of restaurants or cafas nearby."[9]

Y para el tópico "**room**":

- Positivo: "We only stayed for two nights but the room was spacious and clean."
- Negativo: "The room seemed clean at first look, but actually there were stains on the bed covers and the mattress was stained and nasty."

Al analizar todos los tópicos de todos los hoteles tenidos en cuenta, hemos podido ver que existe contradicción en todos ellos.

Esto se debe, principalmente, a que cada viajero tiene una experiencia distinta y la subjetividad está muy presente. Sin embargo, también existe la posibilidad de que se hayan introducido comentarios negativos para desprestigiar deliberadamente a un hotel.

Por ello, hemos analizado la población de tópicos contradictorios dados un hotel y un tipo de usuario específico, es decir, cuántos tópicos contradictorios hay con respecto a la totalidad de los tópicos detectados, lo cual es visible en la siguiente tabla:

[9] En este comentario, extraído directamente del usuario, se puede comprobar que, pese a haber utilizado un corrector ortográfico, todavía pueden existir errores ortográficos, por ejemplo "cafas" en vez de "cafés".

Hoteles\Tipo de usuario	Amigos	Familia	Negocios	Pareja	Solo
Black hotel	17.45 %	19.30 %	12.12 %	25.27 %	11.95 %
Comfort Hotel Bolivar	10.60 %	21.23 %	17.78 %	22.19 %	14.51 %
Excel Hotel Roma Montemario	15.54 %	20.86 %	13.18 %	23.77 %	15.11 %
Grand Hotel Fleming	14.19 %	13.63 %	2.70 %	15.98 %	9.30 %
Hotel Columbus	19.23 %	19.35 %	14.28 %	26.67 %	12.14 %
Hotel Lord Byron	14.52 %	18.67 %	20.89 %	23.97 %	12.35 %
Hotel San Remo	15.87 %	19.77 %	8.77 %	23.09 %	16.46 %
Hotel Turner	13.70 %	18.31 %	13.13 %	27.27 %	11.95 %
Quality Hotel Nova Domus	18.84 %	19.86 %	6.89 %	23.80 %	9.09 %
Sina Bernini Bristol	18.81 %	20.77 %	11.98 %	20.76 %	11.71 %

Tabla 5.6.: Población de tópicos contradictorios por hotel y tipo de usuario

Los porcentajes mostrados en la Tabla 5.6 se han calculado siguiendo la siguiente fórmula:

$$\frac{T\acute{o}picos\ contradictorios * 100}{T\acute{o}picos\ totales}$$

También hemos realizado un análisis para comprobar si existía contradicción entre tópicos iguales de distintos tipos de usuario para un hotel específico, obteniendo la siguiente tabla como resultado para todos los hoteles.

Tipo de usuario	Amigos	Familia	Negocios	Pareja	Solo
Amigos	-	-	-	-	-
Familia	Si	-	-	-	-
Negocios	Si	Si	-	-	-
Pareja	Si	Si	Si	-	-
Solo	Si	Si	Si	Si	-

Tabla 5.7.: Existencia de contradicción entre distintos tipos de usuario para un mismo hotel

Para cada par de tipos de usuario comparados realizamos el siguiente proceso para detectar si hay contradicción entre ellos:

1. Para cada tópico, calculamos el número de frases positivas y negativas existentes y guardamos qué conjunto es mayor y cuál es menor para cada tipo de usuario.

   Para facilitar la comparación, guardaremos en las variables "mayor" y "menor" una cadena que indique cuál de los conjuntos es el mayor y cuál el menor.

   Del mismo modo, guardaremos también si alguno de los dos conjuntos es cero o si los conjuntos son iguales, como se indica en la ecuación 5.2 mostrada a continua-

ción:

$$mayor/menor = \begin{cases} Positivo & \text{si } Positivos < Negativos \\ Negativo & \text{si } Positivos > Negativos \\ Igual & \text{si } Positivos = Negativos \\ Cero & \text{si } 0 \end{cases} \tag{5.2}$$

2. Comparamos las variables mayor y menor de los dos tipos de usuario, devolviendo un "**Si**" en el caso de que ambas sean diferentes entre sí.

Variable	**Usu A**	**Usu B**	**Resultado**
mayor	positivo	negativo	**Si**
menor	negativo	positivo	

Variable	**Usu A**	**Usu B**	**Resultado**
mayor	positivo	positivo	**No**
menor	negativo	negativo	

Variable	**Usu A**	**Usu B**	**Resultado**
mayor	negativo	positivo	**Si**
menor	positivo	negativo	

Variable	**Usu A**	**Usu B**	**Resultado**
mayor	negativo	negativo	**No**
menor	positivo	positivo	

Tabla 5.8.: Tablas de clasificación para la comparación de tópicos entre distintos tipos de usuario de un mismo hotel

En los casos en los que ambas son "**Igual**", o alguna de las 2 es "**Cero**" se devolverá un "**No**" como resultado.

Debido a que no es posible conocer, a priori, el número de elementos del mismo tópico que habrá entre distintas polaridades, se puede dar el caso de que un tipo de usuario tenga 7 frases positivas y una negativa para un tópico dado, mientras que otro distinto tenga 300 frases positivas y 10 negativas para el mismo tópico, a lo cual nos devolverá que NO hay contradicción al hacer la comparación de las cadenas "positivo" y "negativo" y ver que para los dos tipos de usuario el conjunto de frases positivas es mayor que el de las negativos.

Por ejemplo, para un tópico "X":

Polaridad	**Familia**	**Amigos**	**Resultado**
Positiva	7 (mayor)	300 (mayor)	**No**
Negativa	1 (menor)	10 (menor)	

En el caso de que el conjunto de frases positivas y el de las negativas para un mismo tópico sean iguales en tamaño, se clasificarán como "iguales" ambos conjuntos, por lo que se puede dar que para un tipo de usuario exista un tópico con 200 positivos y 200 negativos, y otro tipo de usuario en el que el mismo tópico tenga 3 positivos y 3 negativos, clasificando dichos conjuntos como "iguales" y devolviéndonos un NO en la contradicción para ese tópico y buscando si existe contradicción en otro tópico para ese tipo de usuario.

Se debe tener en cuenta que en nuestra propuesta estamos dando prioridad a la polaridad de las opiniones que más abundan, pero existe la posibilidad de que no

se detecte contradicción cuando en realidad sí que la hay. Esto es una mejora a abordar como trabajo futuro del TFG.

Tomando los mismos tipos de usuario utilizados en el ejemplo anterior, un tópico "Y" mostraría lo siguiente:

Polaridad	Familia	Amigos	Resultado
Positiva	200 (iguales)	7 (iguales)	**No**
Negativa	200 (iguales)	7 (iguales)	

También se puede encontrar la lista de los 506 tópicos[10] contradictorios que aparecen como mínimo en uno de los tipos de usuario existentes en los 10 hoteles seleccionados.

Una vez hecho todo esto podemos pasar a la interpretación de los datos obtenidos.

5.1.4. Interpretación

En esta sección hablaremos sobre cómo se interpretan los datos que hemos obtenido tras la fase de "Identificación del tópico" dentro de nuestro sistema.

5.1.4.1. Ranking de relevancia

La necesidad de priorizar la información relevante existente dentro de un conjunto de datos es importante para minimizar la cantidad de esfuerzo invertido y maximizar la utilidad del mismo.

Por eso, en esta sección comentaremos el método que se ha utilizado para ordenar los datos obtenidos de la fase de "Identificación del tópico" y cómo lo hemos utilizado para generar un conjunto ordenado por relevancia.

Como hemos mencionado en la sección 5.1.2.2, se ha realizado un cálculo de la relevancia del autor.

Este número nos da una vista previa de cuán relevante es un comentario con respecto a los otros, ya sea porque el comentario puede contener la información más actual al haberse publicado antes, como por la misma reputación del autor, basada en el conjunto de publicaciones que ha realizado el mismo y cuántas de ellas han sido consideradas útiles por otras personas.

Por otra parte, se realizó un análisis sobre qué tópicos eran más relevantes y cuales no para un tipo específico de usuario.

Para ello, nos basamos en el número de apariciones de un tópico para un tipo específico de usuario en diferentes hoteles.

Esto es que, si, por ejemplo, el tópico "bar" aparece en, al menos, una frase de dos o más hoteles distintos, como pueden ser los hoteles "Sina Bernini Bristol" y "Quality Hotel Nova Domus", para un tipo de usuario específico como "Negocios", este tópico se considerará relevante.

Con esto se elaboraron 6 listas con los tópicos más relevantes para cada uno de los tipos de usuario individuales y para una categoría general, que aunaba todos las anteriores.

[10] `https://goo.gl/5LCCYU`

Junto a esto, se realizó un proceso de clusterización de los tópicos, consistente en la agrupación de palabras relacionadas con los tópicos relevantes en clústeres, o grupos de palabras.

Para entenderlo de manera más sencilla, si imaginas una nube de puntos, los grupos de puntos que se encuentran más próximos unos de otros serían considerados clústeres.

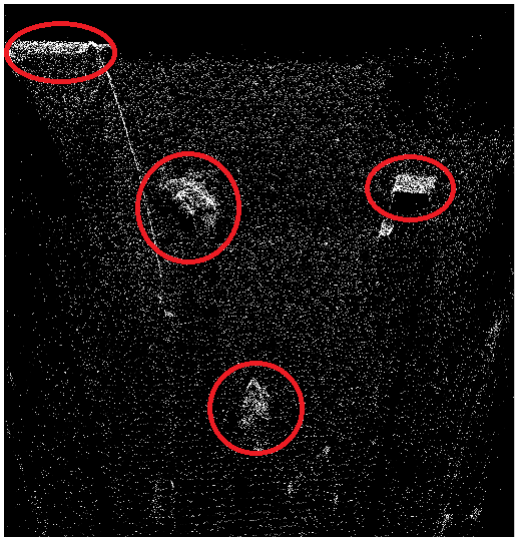

Figura 5.3.: Representación gráfica de clústeres en una nube de puntos

En la Figura 5.3, los círculos rojos representan un clúster. Si cada pequeño punto de la imagen anterior fuera una palabra, los más próximos entre sí serían palabras relacionadas como, por ejemplo "cookie", "milk" y "breakfast" serían palabras relacionadas.

Con estos clústeres obtenemos una lista de palabras relacionadas como la que se muestra a continuación:

```
bar:['cafe', 'restaurant', 'pub', 'club', 'bars', 'party', 'hotel', 'lounge', 'pit
    ', 'house']
bartender:['waitress', 'waiter', 'barman', 'bartenders', 'barista', 'cocktail', '
    librarian', 'asks', 'whiskey', 'alcoholic']
base:['bases', 'force', 'a', 'top', 'point', 'social', 'ca', 'as', 'standard', '
    central']
basic:['flats', 'white', 'basics', 'example', 'ratchet', 'black', 'dress', 'these
    ', 'bitches', 'stupid']
basilica:['cathedral', 'vatican', 'duomo', 'parroquia', 'catedral', 'borgo', '
    ermita', 'monastery', 'santuario']
```

A la izquierda de los dos puntos podemos ver la palabra base y, a la derecha, la lista (representada en Python con corchetes "[ ]") de tópicos relacionados con éste, separados por comas.

El código que realiza este proceso se puede encontrar en el apartado A.7 del anexo.

Una vez obtenidas la lista de tópicos relevantes y la de tópicos clusterizados, seleccionaremos, de la lista de tópicos clusterizados, aquellos que se encuentran dentro de la lista de tópicos relevantes, de modo que podamos conocer si los tópicos de nuestras frases son relevantes o están relacionados con uno.

Todo esto nos es útil, ya que utilizaremos esta información para actualizar la puntuación básica de la reputación de un autor basándonos en la relevancia del tópico teniendo en cuenta el usuario que lo usa.

Para actualizar la puntuación se ha seguido la heurística visible en la ecuación 5.3:

$$\textit{Puntuación de relevancia del tópico} = \begin{cases} +10 & \text{si } \textit{tópico relevante} \\ +5 & \text{si } \textit{tópico relacionado con relevante} \\ -5 & \text{si } \textit{tópico no relevante} \\ \pm 0 & \text{si } \nexists \textit{ tópico} \end{cases} \tag{5.3}$$

Entendemos que un tópico relacionado con uno relevante es aquel que, según la lista de palabras relacionadas, se relacionan entre sí sin estar dentro de la lista de tópicos relevantes.

Por ejemplo, en el ejemplo anterior la palabra "basilica" sería un tópico relevante, mientras que las palabras "cathedral" , "vatican" , etc. serían tópicos relacionados con el relevante.

Al final de todo esto obtendremos que las puntuaciones son el resultado de la fórmula 5.4:

$$\textit{Puntuación} = \begin{matrix} \textit{Puntuación de relevancia del autor} \\ + \\ \textit{Puntuación de relevancia del tópico} \end{matrix} \tag{5.4}$$

La obtención de la "puntuación de relevancia del autor" mencionada en la fórmula anterior se ha explicado en la sección 5.1.2.2 de este TFG.

Una vez obtenidas las puntuaciones, ordenamos las frases de mayor a menor, teniendo en cuenta que sus valores oscilarán en el rango [-3,26] debido a las diferentes heurísticas involucradas en su cálculo.

5.1.4.2. Selección de frases

En esta sección hablaremos sobre qué criterios hemos utilizado para seleccionar las frases que se utilizarán en el resumen final.

Dependiendo de la sección del resumen en la que nos encontremos, las cuales explicaremos en la sección 5.1.5, la selección de frases debe variar.

Algunas partes utilizan la totalidad de las frases para sacar estadísticas sobre los tópicos, como el porcentaje de cuántos de ellos expresan opiniones positivas y cuántos negativas.

Otras hacen selecciones de las frases con mayor aparición basándose en el sentimiento que muestran, mientras que hay otras que devuelven las frases de los 3 tópicos relevantes más comentados, así como el porcentaje de aparición de estos.

Por último, hay una parte concreta que selecciona frases contradictorias dentro del corpus.

Todas estas selecciones son utilizadas en las diferentes plantillas definidas para cada una de las partes del resumen, las cuales explicamos a continuación.

5.1.5. Generación del resumen

En esta última fase, utilizamos la información recolectada en la fase de Interpretación para montar un nuevo resumen basándonos en plantillas.

5.1.5.1. Plantillas

La última parte de la generación de resúmenes consiste en utilizar la información recolectada para crear un nuevo texto que sea coherente y proporcione la información de un modo claro y conciso.

Para ello hacemos uso de varias plantillas, separadas por secciones, que se montarán una tras otra para la obtención del resumen final.

En este TFG se decidió separar las plantillas en 6 partes listadas a continuación:

- Frase de introducción
- Opiniones positivas sobre el hotel
- Opiniones negativas sobre el hotel
- Aspectos más comentados del hotel
- Diversidad de opinión (Contradicción)
- Opiniones sobre aspectos específicos (Metadatos)

La primera parte del resumen consiste en una frase de introducción que nos presenta el nombre del hotel y la opinión general del hotel.

Seguidamente se introduce el tópico más comentado entre los positivos, así como algunos ejemplos del mismo.

Del mismo modo, se muestra el tópico negativo con mayor aparición junto con frases en las que se ha detectado este tópico.

Tras haber introducido, respectivamente, los tópicos positivo y negativo más comentados para el hotel y el tipo de usuario elegido, se comentan el número de frases analizadas y los tópicos detectados, así como los 3 tópicos más comentados entre todos los existentes, mencionando también el porcentaje de su aparición en la totalidad de las frases analizadas.

Seguidamente introducimos la diversidad de opiniones existente en los datos, es decir, introducimos la contradicción que existe entre frases que hablan sobre el mismo tópico para un hotel específico. Para ello, hacemos uso de algunos ejemplos, mencionando también el porcentaje de tópicos contradictorios que hay en el corpus.

Por último, mostramos una lista de los aspectos específicos de un hotel, proporcionados por TripAdvisor, que han votado los usuarios, mostrando la puntuación media de entre todos los usuarios que han votado.

En total se crearon 17 plantillas que, combinadas, formarían un resumen adaptado a un tipo de usuario específico.

A continuación mostramos un ejemplo real de un resumen final generado por nuestro sistema.

Listado 5.2: Ejemplo de resumen generado

```
The solo travelers who stayed at the quality hotel nova domus had a generally good
     opinion of this hotel, as the 61.33% of the comments are positive.
The users seem comfortable talking about "breakfast" because they express
    positiveness on the 75.0% of the comments related to this topic. For example,
    you can see that on comments like "The wifi was sketchy but the free breakfast
     kinda made up for it." or "My room was clean the staff was friendly and the
    breakfast was good.".
On the other hand the customers tend to be negative when talking about "hotel" as
    you can see on the 57.14% of the comments. Proof of that are comments like "
    Now the Hotel could use a little paint." and "If there was a way to rate the
    Staff and actual hotel separately they indeed would get a 5 Star.".
We can also see that, from the 75 reviews crawled for this type of traveler and
    the 5 topics detected, the most commented aspects for this hotel were the "
    hotel", noticeable on the 9.33% of the comments, "breakfast" with a 5.33% of
    appearance, and "stay" with a 2.67%.
This type of traveller has shown different opinions about some topics like "
    breakfast" and "hotel". For the topic "breakfast" they comment "The breakfast
    was ok." and also "If your not used to a European breakfast it takes a little
    to get used to it.". The same occurs for the 10.71% of the previously
    commented topics on this hotel, so you cannot establish a definitive
    conclusion about them.
Finally, we list below the average scores of some especific aspects of the hotel
    that have been ranked by the users:
Service: 3/5
Cleanliness: 3/5
Value: 3/5
Sleep Quality: 3/5
Location: 3/5
Rooms: 3/5
```

Tanto las plantillas como los métodos utilizados para la interpretación de los datos se pueden encontrar en el apartado A.8 del anexo.

En el siguiente capítulo comentaremos la evaluación que se ha realizado y los resultados que hemos obtenido de ella.

6. Evaluación y Resultados

En este capítulo hablaremos sobre la metodología que hemos llevado a cabo para la evaluación y los resultados que se han obtenido de esta.

6.1. Metodología de evaluación

Teniendo en cuenta de que el sistema propuesto genera automáticamente resúmenes a partir de información, hemos evaluado distintas metodologías de evaluación.

Debido a la gran cantidad de ambigüedad que introducen las metodologías basadas en una evaluación automática o semi-automática en lo que respecta a fiabilidad de la evaluación, así como la falta de sistemas existentes que puedan evaluar este tipo de resúmenes, dichas metodologías fueron descartadas.

Por ello, se decidió darle un enfoque manual a la evaluación del sistema, eligiendo a evaluadores con conocimientos en Inglés para que leyesen y evaluasen aspectos varios del resumen.

En un principio se evaluó el uso de herramientas como Crowdflower, ahora llamada FigureEight[1], sin embargo, debido a la falta de tiempo y la extensión de la tarea, se decidió enviar mensajes a distintos grupos y perfiles de personas (investigadores, estudiantes, compañeros) que tuvieran un alto conocimiento en Inglés para que evaluaran los textos.

A estas personas únicamente se les informó sobre el hecho de que tenían que evaluar unos textos, sin saber en ningún momento la procedencia de cada uno de ellos.

Se preparó un formulario hecho con la herramienta de formularios que proporciona Google Drive para la evaluación.

En cada formulario se le daba al evaluador un resumen de uno de los hoteles del corpus para leer y una serie de preguntas escritas en Inglés y Español que debía responder tras haber leído el resumen.

La gran mayoría de las preguntas se respondieron siguiendo la escala de Likert[2], basada en una puntuación del 1 al 5, donde 1 es totalmente de acuerdo y 5 totalmente en desacuerdo.

En las Figuras 6.1 y 6.2, visibles a continuación, mostramos un ejemplo del formulario utilizado en la evaluación.

[1] `https://www.figure-eight.com/`
[2] `https://es.wikipedia.org/wiki/Escala_Likert`

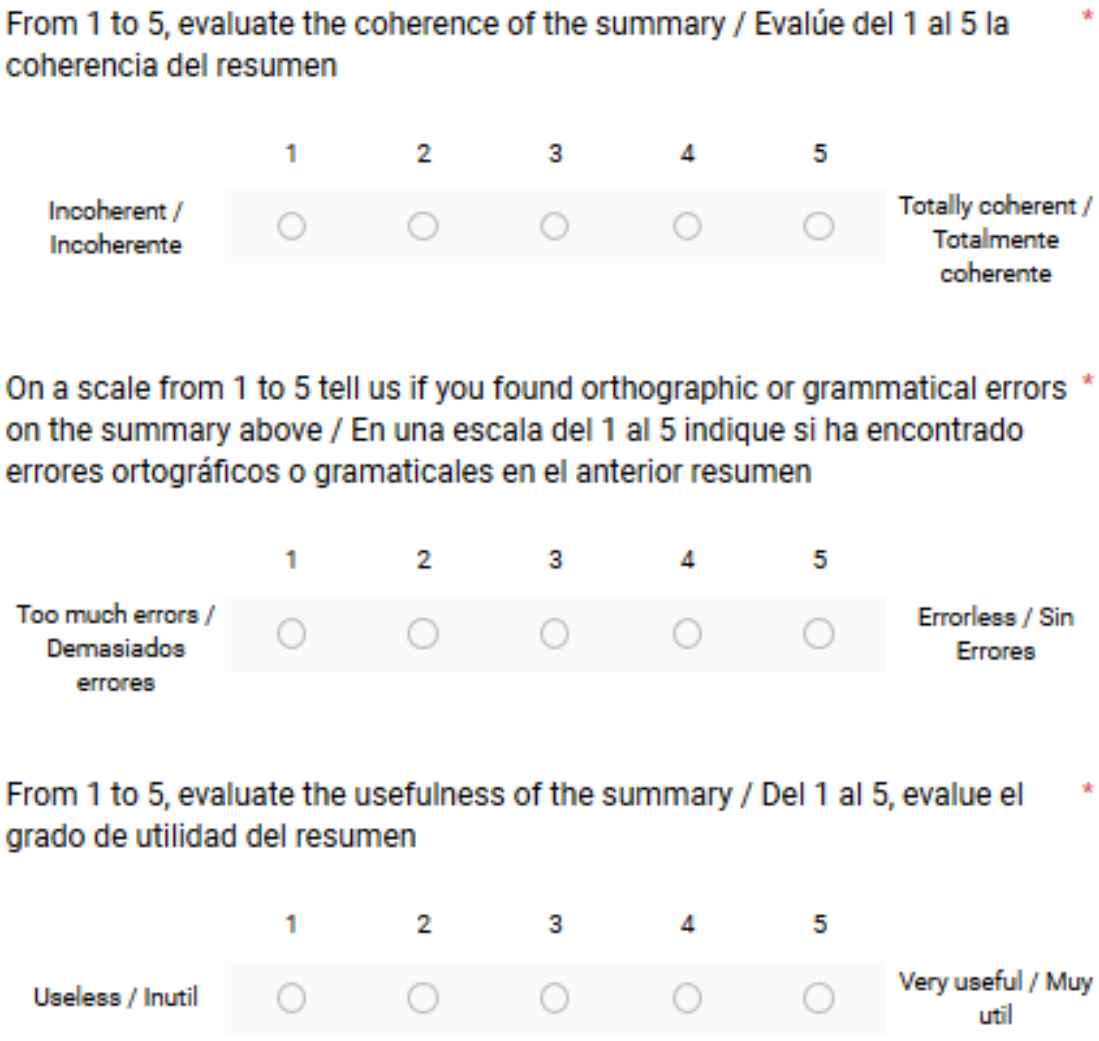

Figura 6.1.: Formulario de evaluación (Parte 1)

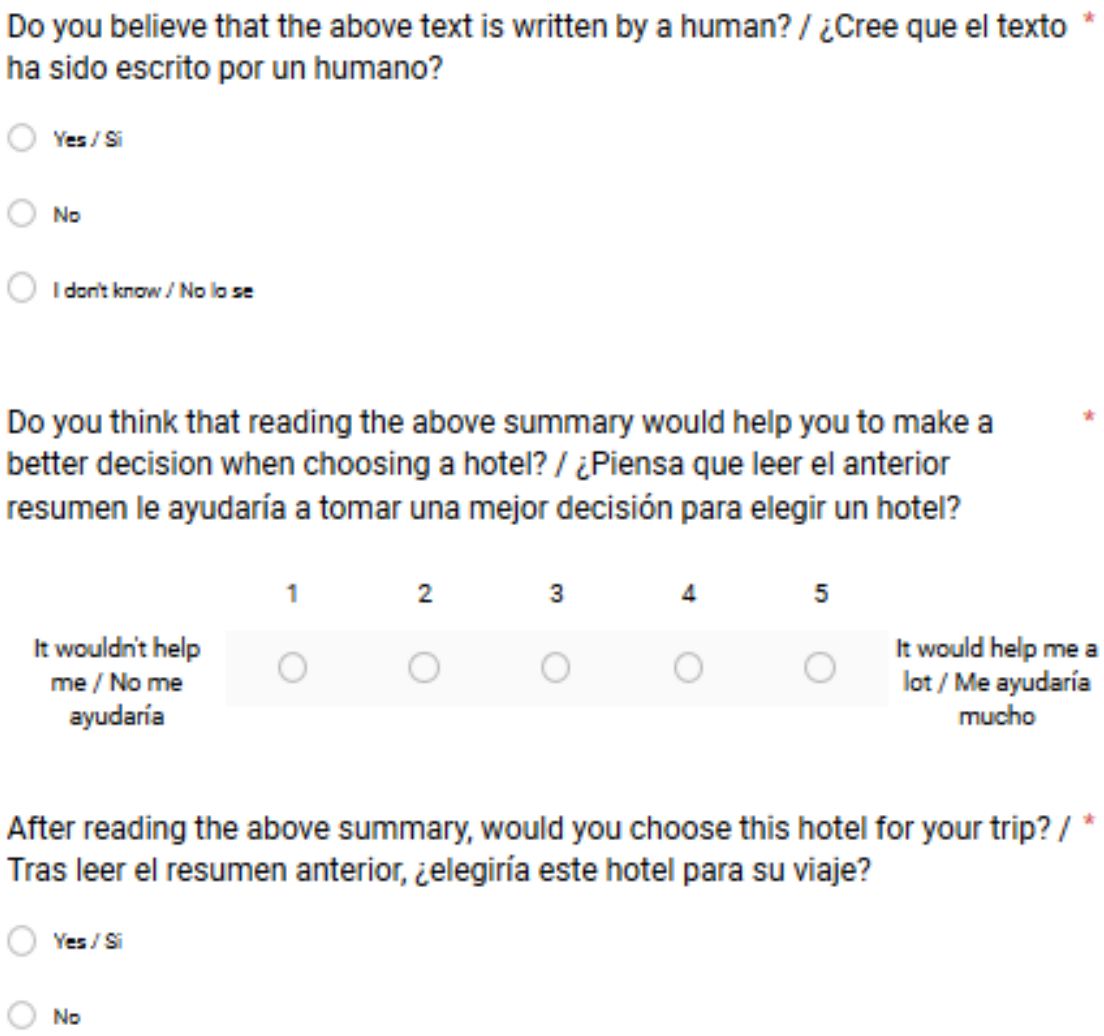

Figura 6.2.: Formulario de evaluación (Parte 2)

Además de esto se evaluaron, junto con el sistema que se propone en este TFG, varios sistemas disponibles de resúmenes genéricos que se utilizaron para comparar nuestro sistema con otros y ver si los evaluadores elegirían antes nuestro sistema frente a uno genérico.

6.1.1. Comparación con otros sistemas

Entre los distintos sistemas genéricos de generación de resúmenes existentes, se escogieron 10 sistemas en total, 5 online y 5 locales, para la generación de resúmenes haciendo uso del corpus recolectado. Este corpus se utilizó para la generación de resúmenes en todos los sistemas, tanto en el nuestro como en los mencionados anteriormente en la sección 2.3.

A la hora de evaluarlos se utilizó el mismo formulario que en nuestro sistema, para así poder compararlos más equitativamente.

También, y del mismo modo que a la hora de evaluar nuestro sistema, se proporcionó a cada evaluador los resúmenes generados para un hotel específico, intercalándolos con las evaluaciones de nuestro sistema.

Para evitar que los evaluadores dejaran de tener interés o tuvieran que releer los textos, ya que algunos de ellos son de una extensión considerable, se decidió agrupar los resúmenes en grupos de 3, tras los cuales se realizaría una comparativa entre los mismos.

Para ello, se introdujeron páginas intermedias entre cada grupo de 3 resúmenes, las cuales pedían la opinión de cada usuario sobre los 3 resúmenes anteriormente evaluados de forma individual.

Podemos ver en las Figuras 6.3 y 6.4 un ejemplo de los formularios utilizados para la evaluación.

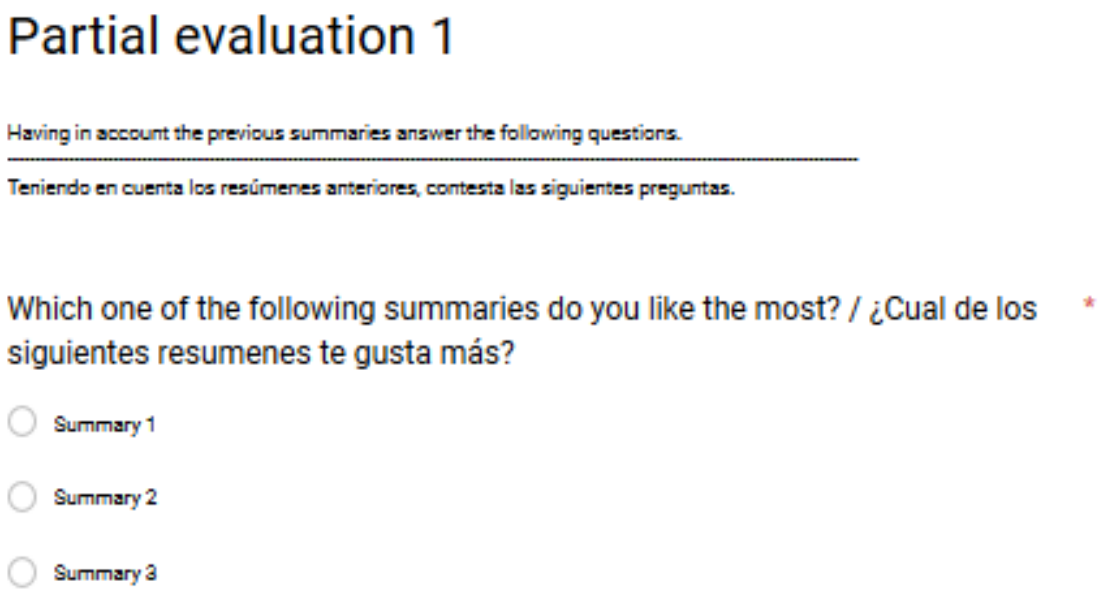

Figura 6.3.: Formulario de evaluación parcial (Parte 1)

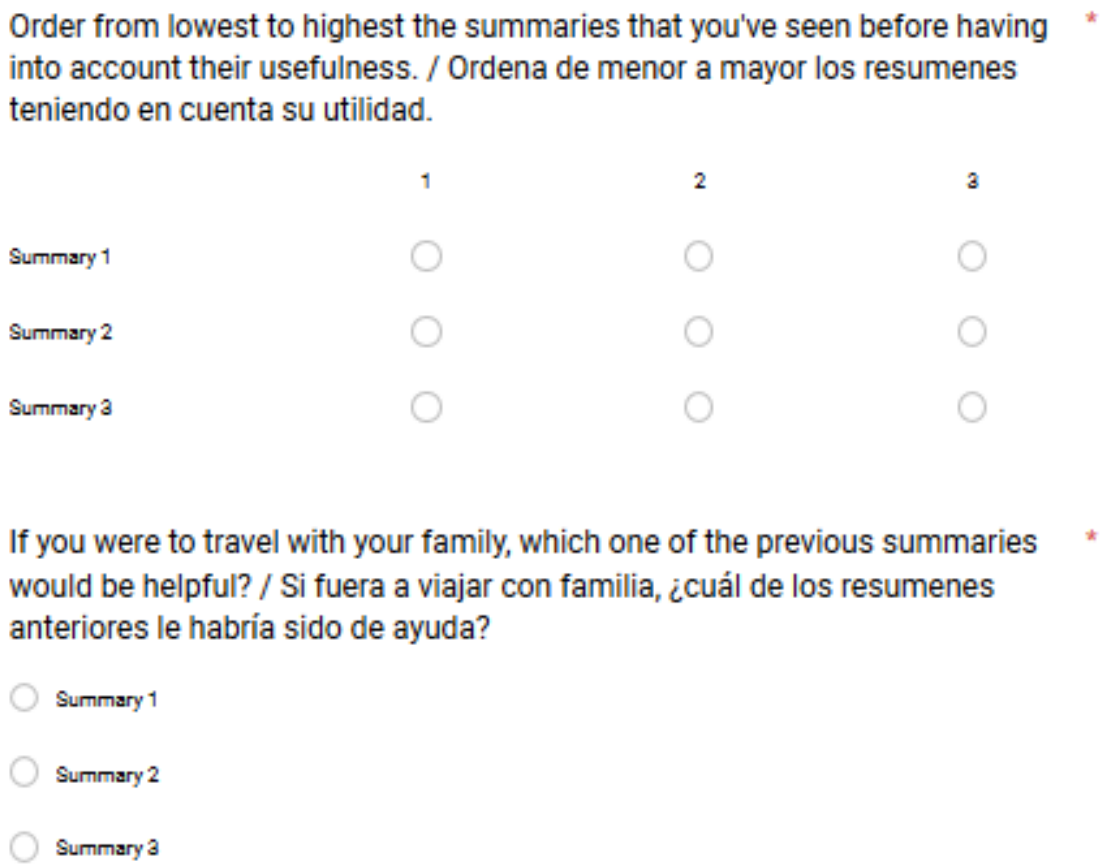

Figura 6.4.: Formulario de evaluación parcial (Parte 2)

Para que cada uno de los grupos mencionados contuvieran 2 resúmenes generados por plataformas genéricas y 1 del sistema propuesto, se decidió un orden de aparición en el formulario.

El orden visible a continuación se debe a la necesidad de que un mismo evaluador leyera y evaluase todos los resúmenes generados por distintas plataformas para un hotel específico. Por esto, se decidió un orden de aparición de los resúmenes basándonos en el sistema que lo generó.

- Preguntas genéricas

- Automatic Text Summarizer
- **Sistema propuesto: Familias**
- Compendium

- *Evaluación Intermedia 1*

- ExplainToMe
- FreeSummarizer
- **Sistema propuesto: Negocios**

- *Evaluación Intermedia 2*

- **Sistema Propuesto: Solo**
- MEAD
- OpenTextSummarizer (OTS)

- *Evaluación Intermedia 3*

- SemPCA-Summarizer
- **Sistema propuesto: Amigos**
- SMMRY

- *Evaluación Intermedia 4*

- SummaNLP
- TextRank
- **Sistema propuesto: Parejas**

- *Evaluación Intermedia 5*

Además, y como se indica en la lista anterior, para conocer el perfil genérico de las personas que han realizado la evaluación, se introdujo al principio del formulario una página de preguntas genéricas visibles en las figuras 6.5 y 6.6.

Tras esto, comienza la evaluación individual de los sistemas y, tras la evaluación de 3 de ellos, una evaluación intermedia que compara los 3 resúmenes que acaba de leer el evaluador, como hemos explicado con anterioridad.

Este proceso de evaluación de 3 resúmenes individuales y 1 comparativo se repite hasta la evaluación de todos los resúmenes generados tanto por nuestro sistema como por el resto de sistemas genéricos.

Es necesario mencionar que en ningún momento se informa al evaluador sobre qué sistema se está evaluando.

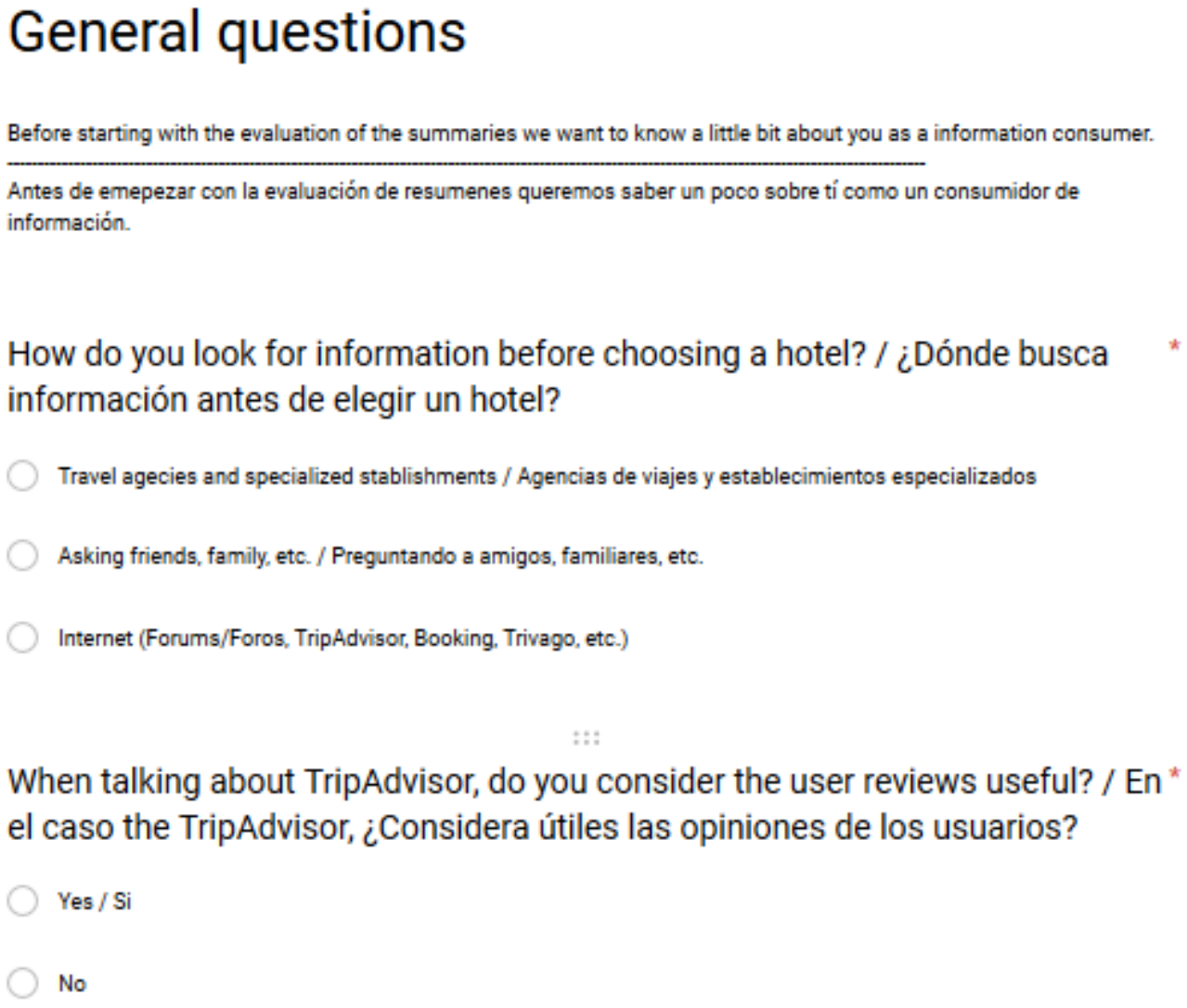

Figura 6.5.: Formulario de preguntas genéricas sobre el evaluador (Parte 1)

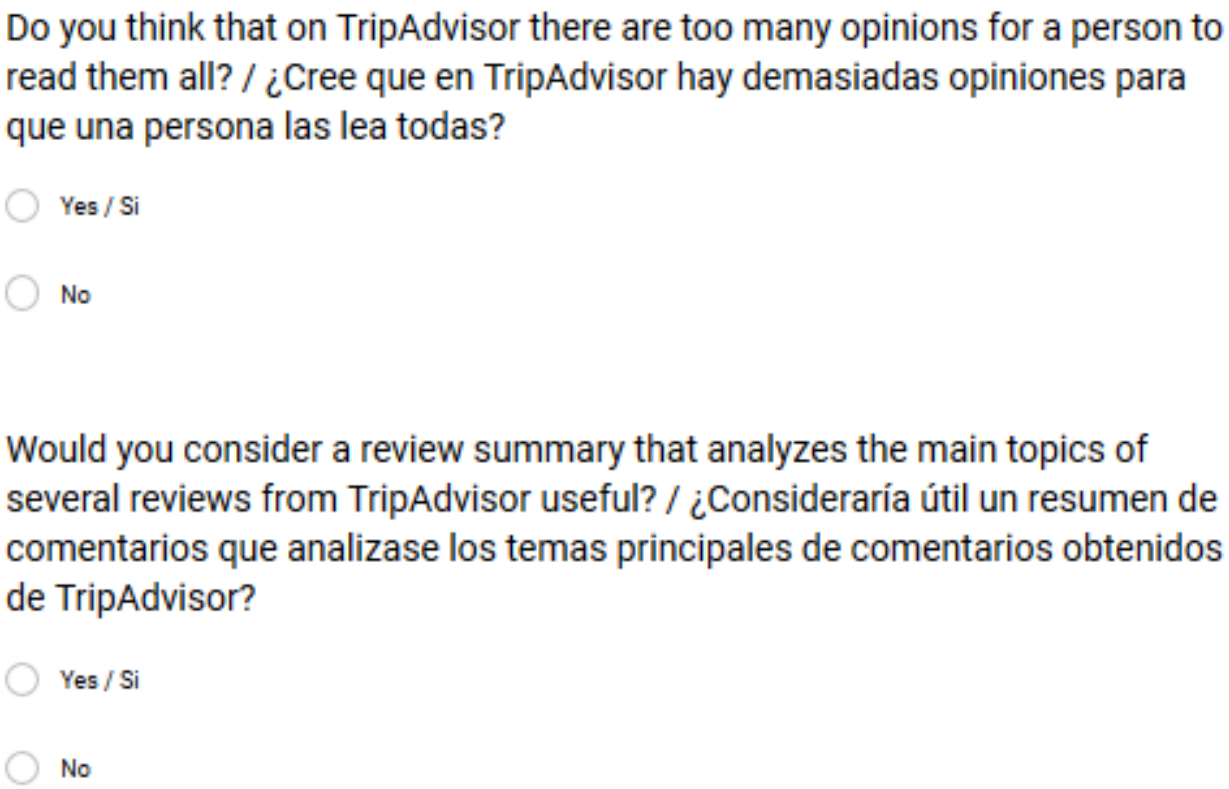

Figura 6.6.: Formulario de preguntas genéricas sobre el evaluador (Parte 2)

En la próxima sección mostraremos y explicaremos los resultados obtenidos de la evaluación explicada hasta el momento.

6.2. Resultados de la evaluación

En esta última sección analizaremos y comentaremos los resultados obtenidos en la evaluación manual realizada.

Según los datos recolectados en la evaluación, podemos observar que la mayoría de los usuarios utilizan Internet para informarse sobre los hoteles en los que se alojarán durante su viaje.

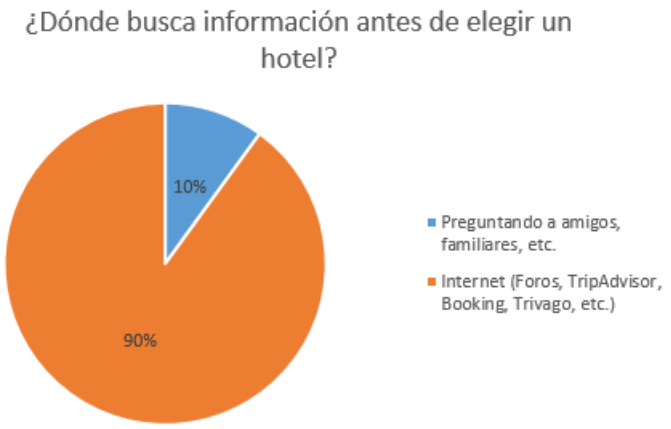

Figura 6.7.: Encuesta sobre métodos de búsqueda comúnmente utilizados por los evaluadores

En la Figura 6.7 únicamente se muestran 2 opciones, sin embargo, en la figura 6.5 se muestra como se les dio a los usuarios 3 opciones entre las que escoger. El hecho de que únicamente aparezcan 2 de las 3 opciones se debe a que ninguno de los evaluadores escogió dicha respuesta a la hora de responder el formulario.

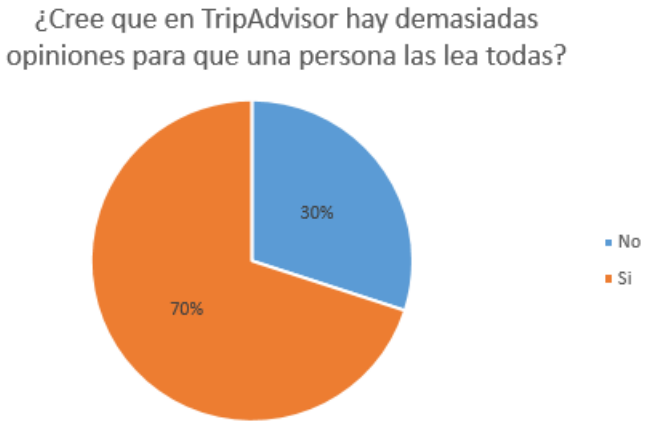

Figura 6.8.: Encuesta sobre la opinión de los evaluadores con respecto a la cantidad de información proporcionada por TripAdvisor

Los evaluadores estuvieron de acuerdo en que las opiniones de TripAdvisor eran útiles, del mismo modo que creían en que sería útil un resumen de estos comentarios. Esto se debe a la grandísima cantidad de datos existente, lo que hace imposible su completa lectura y comprensión.

Tras conocer un poco el perfil de los evaluadores, hicimos que leyeran los resúmenes individualmente y respondieran a las preguntas mostradas en las subsecciones anteriores.

Tras haber leído 3 resúmenes, se les hizo un cuestionario de evaluación intermedio en el que debían comparar dichos 3 resúmenes y seleccionar finalmente con cuál de ellos se quedarían.

Primeramente mostraremos los resultados obtenidos de la evaluación individual de nuestro sistema, tras lo que mostraremos una comparativa con la media obtenida del resto de sistemas.

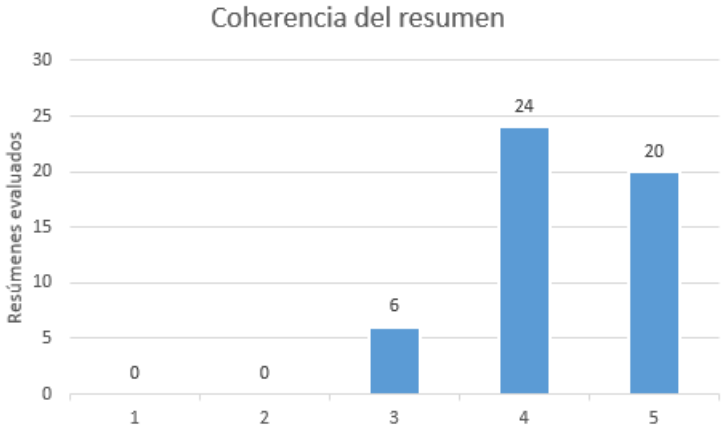

Figura 6.9.: Grado de coherencia en los resúmenes según los evaluadores

La Figura 6.9 nos muestra la puntuación, de menor a mayor que los usuarios han dado a los resúmenes generados para nuestro sistema. Para las estadísticas se han utilizado los 50 resúmenes generados, 5 para cada tipo de usuario de los 10 hoteles seleccionados.

Podemos observar que la mayoría de los resúmenes han sido puntuados con más de un 3 en coherencia, lo que indica que los evaluadores los han encontrado entendibles y con una estructura adecuada.

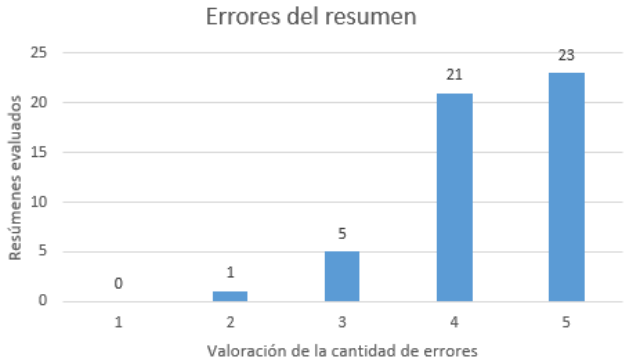

Figura 6.10.: Grado de cantidad de errores según los evaluadores

Por otra parte, en la Figura 6.10 se muestra la evaluación de la cantidad de errores existentes en los resúmenes. A menor cantidad de errores, mejor sería la puntuación dada al resumen en cuestión, como se muestra en la Figura 6.1 que muestra el formulario presentado a los evaluadores.

Del mismo modo que con la coherencia, vemos que la mayoría de los resúmenes han obtenido una buena puntuación con respecto a la cantidad de errores que presentan, lo que ayuda a que los usuarios los comprendan con mayor facilidad.

El hecho de que los resúmenes generados por nuestro sistema tengan pocos errores ortográficos y sean coherentes incrementan su utilidad como resumen, lo que podemos observar en la Figura 6.11 mostrada a continuación.

Figura 6.11.: Grado de utilidad del resumen según los evaluadores

A parte de su utilidad general también preguntamos a los usuarios si los resúmenes les serían útiles a la hora de elegir un hotel para realizar un viaje en específico, a lo cual obtuvimos los resultados mostrados en la Figura 6.12.

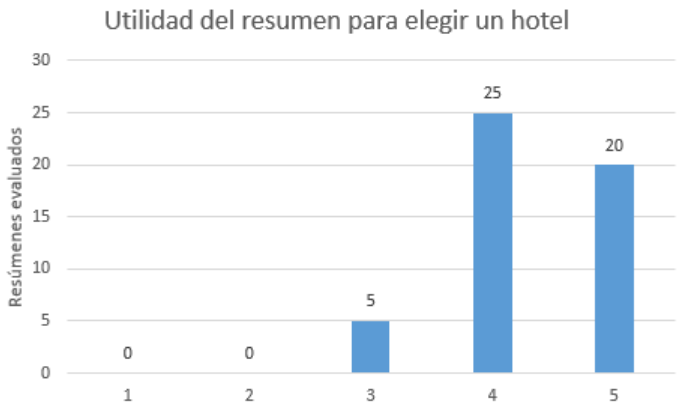

Figura 6.12.: Grado de utilidad del resumen a la hora de elección de un hotel según los evaluadores

Finalmente, les preguntamos a los usuarios su opinión a cerca de si creían que los resúmenes habían sido escritos o no por un humano, obteniendo lo mostrado en la siguiente figura.

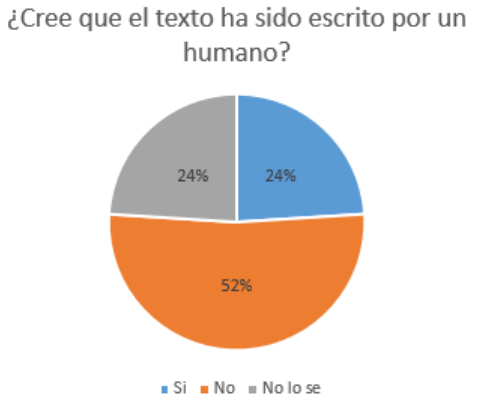

Figura 6.13.: Porcentajes del Test de Turing en la totalidad de los resúmenes

Podemos observar que en un 48 % de los casos, los anotadores no consiguieron distinguir correctamente si el texto había sido escrito por un humano o no.

Al haber en los formularios únicamente 10 resúmenes, incluyendo los creados por nuestro sistema, es difícil que los evaluadores no se den cuenta de cuales son los resúmenes escritos por una plataforma específica. Sin embargo, el resultado obtenido cercano al 50 % es un buen indicio de que, aunque el sistema propuesto deba mejorar en algunos aspectos, en su estado actual ha resultado ser bastante competitivo.

6.2.1. Comparación del sistema propuesto con otros sistemas existentes

Para comprobar la eficiencia del sistema lo hemos comparado con 10 sistemas de generación de resúmenes genéricos, sometiendo estos a las mismas pruebas que al sistema propuesto.

En la Figura 6.14 podemos ver la media de los resultados obtenidos en lo que a coherencia del resumen se refiere.

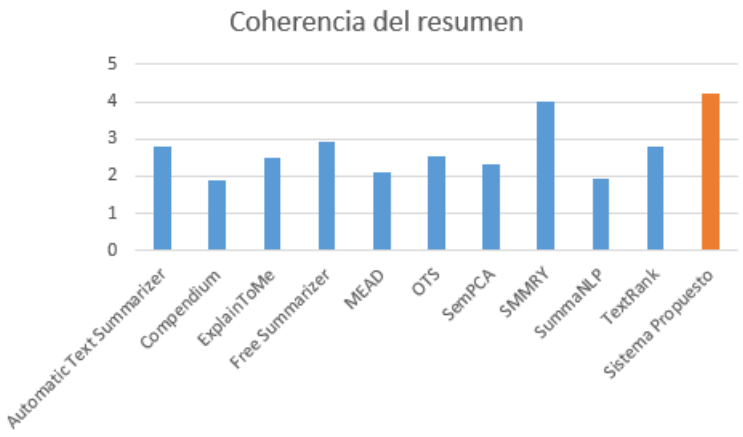

Figura 6.14.: Comparación de coherencia

Observamos que la coherencia de nuestro sistema es ligeramente superior al resto de sistemas genéricos. Esto se debe principalmente a que los sistemas genéricos realizan únicamente una selección de frases, mientras que nuestro sistema analiza más profundamente la información para que el usuario pueda entenderla de manera más sencilla tras sintetizarla.

Por otra parte, también se comprobó la falta de errores ortográficos en los escritos y las medias obtenidas de esas puntuaciones son las siguientes:

Figura 6.15.: Comparación de ausencia de errores ortográficos

Se puede observar claramente que nuestro sistema ha obtenido una puntuación elevada en este apartado y esto se debe en gran parte al proceso de corrección ortográfica del corpus que realiza en sus primeras fases.

En cuanto a la utilidad de los resúmenes se han comparado 2 de ellas, la utilidad del resumen como tal y la utilidad del resumen a la hora de escoger un hotel, cuyos resultados podemos ver en las figuras 6.16 y 6.17.

Figura 6.16.: Comparación de utilidad del resumen

Figura 6.17.: Comparación de utilidad del resumen a la hora de escoger un hotel

En ambas figuras se muestra una clara diferencia entre nuestro sistema y el resto de los sistemas ya que el nuestro muestra resultados por encima del 4 mientras que el resto de sistemas no llegan a dicha puntuación.

Ya que nuestro sistema analiza la información en mayor profundidad y hace uso de los metadatos proporcionados por TripAdvisor para la generación del resumen, los evaluadores han podido ver una mayor utilidad en los resúmenes generados por él.

7. Conclusiones

Gracias a la realización de este TFG he podido aplicar los conocimientos de la carrera, pero también he podido comprobar algunas cosas que todavía no había experimentado debidamente.

Entre algunas de las cosas que he podido experimentar se encuentra el haber investigado el mundo del PLN, viendo las diferentes áreas que abarca, como son la detección de tópicos, polaridad y sentimientos, contradicción, análisis semántico y sintáctico, y muchas otras áreas.

El hecho de haber podido investigar en un laboratorio de investigación, con la ayuda de personas que conocen más en profundidad el campo en cuestión, ha sido de lo más útil y enriquecedor para un estudiante de último año.

También he comprobado que la planificación del tiempo y las tareas es una parte muy importante en el desarrollo de un trabajo de gran envergadura como es un TFG o una investigación.

Del mismo modo he podido ver que la búsqueda y análisis de la información existente en el campo de la investigación en PLN, al igual que la recolección y anotación de un corpus, es una tarea complicada pero que merece la pena.

Aunque el desarrollo de este TFG se haya terminado, todavía quedan muchísimas cosas que se podrían hacer.

En un futuro, el sistema podría incluir Redes Neuronales Recursivas (RNN) para la detección y clasificación de tópicos, así como la detección de la contradicción de los mismos.

Además de poder aplicar el sistema en otros ámbitos fuera del hotelero, este sistema se podría utilizar junto con un sistema de OCR para el resumen de textos manuscritos, o incluso utilizar sistemas de reconocimiento de voz para la obtención de datos por parte de un usuario. A partir de, por ejemplo, una grabación de voz de un seminario, se podría obtener un resumen escrito sobre el contenido del mismo, ahorrando tiempo al usuario a la hora de escoger si es relevante o no para él.

Bibliografía

[Alcón and Lloret, 2015] Alcón, O. and Lloret, E. (2015). Estudio de la influencia de incorporar conocimiento léxico-semántico a la técnica de análisis de componentes principales para la generación de resúmenes multilingües. *Linguamática*, 7(1):53–63.

[Barrios et al., 2016] Barrios, F., López, F., Argerich, L., and Wachenchauzer, R. (2016). Variations of the similarity function of textrank for automated summarization. *CoRR*, abs/1602.03606.

[Bird and Loper, 2004] Bird, S. and Loper, E. (2004). Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[Dom Lachowicz and Rotem, 2008] Dom Lachowicz, M. G. and Rotem, N. (2008). Open text summarizer, version 0.4. 2, libots group.

[Esteban and Lloret, 2017a] Esteban, A. and Lloret, E. (2017a). Propuesta y desarrollo de una aproximación de generación de resúmenes abstractivos multigénero. *Procesamiento del Lenguaje Natural*, 58:53–60.

[Esteban and Lloret, 2017b] Esteban, A. and Lloret, E. (2017b). Travelsum: A spanish summarization application focused on the tourism sector. *Procesamiento del Lenguaje Natural*, 59:159–162.

[Esuli and Sebastiani, 2007] Esuli, A. and Sebastiani, F. (2007). Sentiwordnet: a high-coverage lexical resource for opinion mining. *Evaluation*, 17:1–26.

[Fernández et al., 2015] Fernández, J., Gutiérrez, Y., Gómez, J. M., and Martínez-Barco, P. (2015). Social rankings: análisis visual de sentimientos en redes sociales. *Procesamiento del Lenguaje Natural*, 55:199–202.

[Hovy, 2003] Hovy, E. (2003). Text summarization. In *The Oxford Handbook of Computational Linguistics 2nd edition*. Oxford University Press.

[Hu et al., 2017] Hu, Y.-H., Chen, Y.-L., and Chou, H.-L. (2017). Opinion mining from online hotel reviews–a text summarization approach. *Information Processing & Management*, 53(2):436–449.

[Jurafsky, 2000] Jurafsky, D. (2000). Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*.

[Lloret, 2011] Lloret, E. (2011). *Text summarisation based on human language technologies and its applications*. Universidad de Alicante.

[Lloret, 2016] Lloret, E. (2016). Introducing the key stages for addressing multiperspective summarization. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 321–326. SCITEPRESS-Science and Technology Publications, Lda.

[Manning et al., 2014] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

[Mihalcea and Tarau, 2004] Mihalcea, R. and Tarau, P. (2004). Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*.

[Miller, 1998] Miller, G. (1998). *WordNet: An electronic lexical database*. MIT press.

[Padró and Stanilovsky, 2012] Padró, L. and Stanilovsky, E. (2012). Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Eight International Conference on Language Resources and Evaluation*.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

[Turing, 1950] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

A. Anexo I

A continuación se encuentran los documentos y elementos importantes para el desarrollo de este TFG.

A.1. Código

Listado A.1: CorrectorOrtografico.java

```
1  package correctorortografico;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.FileReader;
8  import java.io.FileWriter;
9  import java.io.IOException;
10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12
13 /**
14  * El corrector ortográfico toma el fichero words.utf-8.txt como base
        para la corrección automática y
15  * no tiene en cuenta si la palabra está separada o no. En caso de
       que la palabra estuviera bien escrita pero no
16  * estuviera correctamente separada, no la detectaría bien y la
       corregiría con la mejor aproximación posible según su criterio.
17  * El diccionario también está falto de jerga y terminos variados
       como pueden ser "WiFi" o "lol", lo que puede hacer que hayan
18  * correcciones no del todo satisfactorias.
19  *
20  * @author Alejandro Reyes Albillar
21  */
22 public class CorrectorOrtografico {
23
24  /**
25   * @param args the command line arguments
26   */
27  public static void main(String[] args) throws FileNotFoundException,
        IOException {
28   CorrectorOrtografico co= new CorrectorOrtografico();
```

```
29      String text="";
30      if(args.length<2){
31       System.err.println("El uso correcto del programa es \"
             nombreprograma [-d directorio][-f fichero]\"");
32       System.exit(0);
33      }
34      File file= new File(args[1]);
35      switch(args[0]){
36       case "-d"://Lee de un directorio
37      if(!file.exists()){//Si el directorio no existe, informa del error
38       System.err.println("La ruta no existe, proporciona una ruta
             correcta y vuelve a intentarlo.");
39      }
40      else{//Si ya existe el directorio lo limpiamos
41       try {
42      String[] arc= file.list();//Lista de directorios
43      for(String s: arc){
44       File read=new File(file.getPath(),s);
45       String[] arch= read.list();//Lista de archivos
46       for(String b:arch){//Eliminamos las posibles correcciones que
             hubiera previamente
47      if(b.contains("_Corrected.txt")){
48       File erase = new File (read.getPath(),b);
49       erase.delete();
50      }
51       }
52
53       for(String a: arch){
54      File leer= new File(read.getPath(),a);
55      BufferedReader br= new BufferedReader(new FileReader(leer));
56      String linea="";
57      while((linea=br.readLine())!=null){
58       if(linea.contains(" | ") || linea.contains("@")){
59        text+=linea+"\n";
60       }
61       else{
62        text+=co.corregir(linea)+"\n";
63       }
64      }
65      br.close();
66      //Sobreescribimos el fichero con el texto corregido
67      String oldPath=leer.getPath();
68      oldPath=  oldPath.substring(0,oldPath.length()-4);
69      String corrected= oldPath+"_Corrected.txt";
70      //System.out.println(corrected);
71      leer= new File(corrected);
72      BufferedWriter bw= new BufferedWriter(new FileWriter(leer));
73      bw.write(text);
74      bw.close();
```

```
75      text="";
76       }
77      }
78       } catch (FileNotFoundException ex) {
79      Logger.getLogger(CorrectorOrtografico.class.getName()).log(Level.
           SEVERE, null, ex);
80       } catch (IOException ex) {
81      Logger.getLogger(CorrectorOrtografico.class.getName()).log(Level.
           SEVERE, null, ex);
82       }
83      }
84      break;
85       case "-f"://Lee un fichero unico
86      BufferedReader br= new BufferedReader(new FileReader(file));
87      String linea="";
88      while((linea=br.readLine())!=null){
89       if(linea.contains(" | ") || linea.contains("@")){
90      text+=linea+"\n";
91       }
92       else{
93      text+=co.corregir(linea)+"\n";
94       }
95      }
96      br.close();
97      //Sobreescribimos el fichero con el texto corregido
98      String oldPath=file.getPath();
99      oldPath=  oldPath.substring(0,oldPath.length()-4);
100     String corrected= oldPath+"_Corrected.txt";
101     //System.out.println(corrected);
102     file= new File(corrected);
103     BufferedWriter bw= new BufferedWriter(new FileWriter(file));
104     bw.write(text);
105     bw.close();
106     break;
107      default:
108     System.err.println("El primer parámetro debe ser \"-d\" o \"-f\"");
109     }
110
111   }
112
113   public String corregir(String text){
114    JazzySpellChecker jazzySpellChecker = new JazzySpellChecker();
115    String line = jazzySpellChecker.getCorrectedLine(text);
116    return line;
117   }
118
119  }
```

Listado A.2: JazzySpellChecker.java

```
1  package correctorortografico;
2
3  import com.swabunga.spell.engine.SpellDictionaryHashMap;
4  import com.swabunga.spell.event.SpellCheckEvent;
5  import com.swabunga.spell.event.SpellCheckListener;
6  import com.swabunga.spell.event.SpellChecker;
7  import com.swabunga.spell.event.StringWordTokenizer;
8  import com.swabunga.spell.event.TeXWordFinder;
9  import com.swabunga.spell.engine.Word;
10
11 import java.io.File;
12 import java.io.FileNotFoundException;
13 import java.io.IOException;
14 import java.util.ArrayList;
15 import java.util.List;
16
17 /**
18  *
19  * @author Alejandro Reyes Albillar
20  */
21 public class JazzySpellChecker implements SpellCheckListener {
22
23  private SpellChecker spellChecker;
24  final private List<String> misspelledWords;
25
26  /**
27   * get a list of misspelled words from the text
28   * @param text
29   * @return
30   */
31  public List<String> getMisspelledWords(String text) {
32   StringWordTokenizer texTok = new StringWordTokenizer(text,
33  new TeXWordFinder());
34   spellChecker.checkSpelling(texTok);
35   return misspelledWords;
36  }
37
38  private static SpellDictionaryHashMap dictionaryHashMap;
39
40  static{
41
42   File dict = new File("src/correctorortografico/words.utf-8.txt");
43   try {
44    dictionaryHashMap = new SpellDictionaryHashMap(dict);
45   } catch (FileNotFoundException e) {
46    e.printStackTrace();
47   } catch (IOException e) {
48    e.printStackTrace();
```

```
49      }
50    }
51
52    private void initialize(){
53      spellChecker = new SpellChecker(dictionaryHashMap);
54      spellChecker.addSpellCheckListener(this);
55    }
56
57
58    public JazzySpellChecker() {
59
60     misspelledWords = new ArrayList<>();
61     initialize();
62    }
63
64    /**
65     * correct the misspelled words in the input String and return the
          result
66     * @param line
67     * @return
68     */
69    public String getCorrectedLine(String line){
70     List<String> misSpelledWords = getMisspelledWords(line);
71
72     for (String misSpelledWord : misSpelledWords){
73      List<String> suggestions = getSuggestions(misSpelledWord);
74      if (suggestions.isEmpty())
75    continue;
76      String bestSuggestion = suggestions.get(0);
77      line = line.replace(misSpelledWord, bestSuggestion);
78     }
79     return line;
80    }
81
82    public String getCorrectedText(String line){
83     StringBuilder builder = new StringBuilder();
84     String[] tempWords = line.split(" ");
85     for (String tempWord : tempWords){
86      if (!spellChecker.isCorrect(tempWord)){
87    List<Word> suggestions = spellChecker.getSuggestions(tempWord, 0);
88    if (suggestions.size() > 0){
89     builder.append(spellChecker.getSuggestions(tempWord, 0).get(0).
          toString());
90    }
91    else
92     builder.append(tempWord);
93      }
94      else {
95    builder.append(tempWord);
```

```
96      }
97      builder.append(" ");
98     }
99     return builder.toString().trim();
100   }
101
102
103   public List<String> getSuggestions(String misspelledWord){
104
105    @SuppressWarnings("unchecked")
106    List<Word> su99esti0ns = spellChecker.getSuggestions(misspelledWord
           , 0);
107    List<String> suggestions = new ArrayList<String>();
108    for (Word suggestion : su99esti0ns){
109     suggestions.add(suggestion.getWord());
110    }
111
112    return suggestions;
113   }
114
115
116   @Override
117   public void spellingError(SpellCheckEvent event) {
118    event.ignoreWord(true);
119    misspelledWords.add(event.getInvalidWord());
120   }
121
122  }
```

Listado A.3: sentimental.py

```
#Code from https://nlpforhackers.io/sentiment-analysis-intro/
#NLTK SentimentAnalyzer

from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag


lemmatizer = WordNetLemmatizer()


def penn_to_wn(tag):
 """
 Convert between the PennTreebank tags to simple Wordnet tags
 """
 if tag.startswith('J'):
  return wn.ADJ
 elif tag.startswith('N'):
  return wn.NOUN
 elif tag.startswith('R'):
  return wn.ADV
 elif tag.startswith('V'):
  return wn.VERB
 return None


def clean_text(text):
 text = text.replace("<br />", " ")
 text = text.decode("utf-8")

 return text


def swn_polarity(text):
 """
 Return a sentiment polarity: 0 = negative, 1 = positive
 """

 sentiment = 0.0
 tokens_count = 0

 text = clean_text(text)


 raw_sentences = sent_tokenize(text)
 for raw_sentence in raw_sentences:
  tagged_sentence = pos_tag(word_tokenize(raw_sentence))
```

```
49
50      for word, tag in tagged_sentence:
51       wn_tag = penn_to_wn(tag)
52       if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV):
53      continue
54
55       lemma = lemmatizer.lemmatize(word, pos=wn_tag)
56       if not lemma:
57      continue
58
59       synsets = wn.synsets(lemma, pos=wn_tag)
60       if not synsets:
61      continue
62
63       # Take the first sense, the most common
64       synset = synsets[0]
65       swn_synset = swn.senti_synset(synset.name())
66
67       sentiment += swn_synset.pos_score() - swn_synset.neg_score()
68       tokens_count += 1
69
70    # judgment call ? Default to positive or negative
71    if not tokens_count:
72     return 0
73
74    # sum greater than 0 => positive sentiment
75    if sentiment >= 0:
76     return 1
77
78    # negative sentiment
79    return 0
```

Listado A.4: topDec.py

```
#Code from https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/
#Toma como entrada un array con el corpus de documentos como un array de strings

from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import gensim
from gensim import corpora

stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

def clean(doc):
  stop_free = " ".join([i for i in doc.lower().split() if i not in stop]) #Eliminamos stopwords
  punc_free = ''.join(ch for ch in stop_free if ch not in exclude) #Eliminamos los signos de puntuacion
  normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split()) #Obtenemos los lemas de cada palabra
  return normalized

def topDec(doc_complete):
  doc_clean = [clean(doc).split() for doc in doc_complete]
  # Creating the term dictionary of our courpus, where every unique term is assigned an index.
  dictionary = corpora.Dictionary(doc_clean)
  # Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.
  doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]

  # Creating the object for LDA model using gensim library
  # Mas info sobre LDA https://es.wikipedia.org/wiki/Latent_Dirichlet_Allocation
  Lda = gensim.models.ldamodel.LdaModel

  try: #Usamos un try catch para que, en caso de que falle porque el corpus no contenga palabras devuelva una cadena vacia
    # Running and Trainign LDA model on the document term matrix.
    ldamodel = Lda(doc_term_matrix, num_topics=3, id2word = dictionary, passes=50)
    return ldamodel.print_topics()
  except:
    return ""
```

Listado A.5: MultiPerspectiveSummarizer.py (PoS Tagger y Detección de tópico)

```
##################################
#POS Tagger/ANALISIS MORFOLOGICO#
##################################

def NLTKPOSTagger(text):
text= nltk.word_tokenize(text)
return nltk.pos_tag(text)

#####################
#DETECCION DE TOPICO#
#####################

#Llama a la funcion de deteccion de topico del modulo topDec
def topicDetection(corpus):
  return topDec.topDec(corpus)

#Actualiza los topicos en la BBDD con la opcion --topic
def topicos(result):
  topicsList=defaultdict(lambda:0)
  if result:
    comentario=[]
    #ncomentario=int(result[0][1])
    for x in result:
      comentario.append(x[1])
      topics=topicDetection(comentario)#Aqui se guardan los topics
      #Contamos
      topicsList.clear()
      for l in topics:
        aux=re.findall('[A-Za-z]+',l[1])#Obtenemos unicamente el
           topico, eliminando el resto de elementos que no nos
           interesan (de momento)
        for b in aux:
          if(len(aux)>=1):
            topicsList[b]+=1
      comentario=[]
      topicsList2=sorted(topicsList.items(),key=operator.
         itemgetter(1),reverse=True)
      meh=""
      for l in topicsList2: #Transformamos los topicos en string
         para sacar el postagger
        meh+=l[0]+" "
      topicsList2=NLTKPOSTagger(meh) #Analizamos morfológicamente
         la frase
      sublista=[]
      for l in topicsList2:
        if(l[1]=="NN"): #Si es un sustantivo se añadirá al
           conjunto final en el mismo orden en el que se leen, de
           mayor a menor relevancia
```

```
42              sublista.append(l[0])
43          sublista=sorted(sublista)
44          topic=""
45          if (len(sublista)>=1):
46            topic=sublista[0]
47          updateTopic(x[0],topic) #Actualiza el tópico para la frase
               en la BBDD
```

Listado A.6: createDatabase.sh

```
#! /bin/bash

newUser='ara65' #Usuario local
newDbPassword='ara1995' #Contraseña de acceso a la BBDD

host=localhost #DO NOT TOUCH IF POSSIBLE
newDb='MGS'

##############################
# DO NOT TOUCH UNDER ANY CASE#
##############################

commands="DROP USER IF EXISTS ${newUser}@${host};DROP DATABASE IF EXISTS ${newDb};
    CREATE DATABASE ${newDb};CREATE USER '${newUser}'@'${host}' IDENTIFIED BY '${
    newDbPassword}';GRANT USAGE ON *.* TO '${newUser}'@'${host}' IDENTIFIED BY '${
    newDbPassword}';GRANT ALL privileges ON ${newDb}.* TO '${newUser}'@'${host}';
    FLUSH PRIVILEGES;USE ${newDb};CREATE TABLE Hotel (nombre VARCHAR(100),PRIMARY
    KEY(nombre)) ENGINE=InnoDB;CREATE TABLE Comentario (id INT,titulo TEXT,
    scoreHotel INT, userType VARCHAR (8), nacionalidad VARCHAR(100), fecha VARCHAR
    (50), score INT, hotel VARCHAR(100), PRIMARY KEY (id), FOREIGN KEY (Hotel)
    REFERENCES Hotel(nombre) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;
    CREATE TABLE Frase (id INT, frase TEXT, topico TEXT, comentario INT, PRIMARY
    KEY (id), FOREIGN KEY (comentario) REFERENCES Comentario(id) ON DELETE CASCADE
     ON UPDATE CASCADE) ENGINE=InnoDB;CREATE TABLE Metadato (id INT, meta VARCHAR
    (100), metaScore INT, id_comentario INT, PRIMARY KEY (id), FOREIGN KEY (
    id_comentario) REFERENCES Comentario(id) ON DELETE CASCADE ON UPDATE CASCADE)
    ENGINE=InnoDB;"

echo "${commands}" | /usr/bin/mysql -u root -p
```

Listado A.7: clustering.py

```
1 #Code from https://github.com/darenr/wordnet-clusters/blob/master/
     wv.py
2 #Slightly modified by Alejandro Reyes Albillar 13/03/2018 17:40
     following the manual on https://radimrehurek.com/gensim/scripts
     /glove2word2vec.html
3 import os
4 import json
5 from gensim.models import Word2Vec
6 from gensim.models import KeyedVectors
7 from gensim.test.utils import datapath, get_tmpfile
8 import sys
9 import os.path
```

```
10
11 print ' *', 'loading wv model'
12
13 path=os.environ['HOME'] + "/models/"
14
15 #Seleccionamos el corpus del cual obtendremos el cluster
16 #name="glove.6B.300d.txt"
17 #name="glove.42B.300d.txt"
18 name="glove.twitter.27B.200d.txt"
19 tmp_file = get_tmpfile(path+"w2v."+name)
20
21 if not os.path.exists(path+"w2v."+name):
22   modelFile = path + name
23   glove_file = datapath(modelFile)
24   from gensim.scripts.glove2word2vec import glove2word2vec
25   glove2word2vec(glove_file, tmp_file)
26
27 model = KeyedVectors.load_word2vec_format(tmp_file, binary=False)
     #Changed because of deprecation on "model = Word2Vec.
     load_word2vec_format(modelFile, binary=False)""
28 print ' *', 'model ready'
29
30 print(model)
31 lista=[]
32 for l in open(sys.argv[1]):
33   lista.append(l.split("\n")[0])
34 words=set(lista)
35
36 results = {}
37 unmatched = []
38
39 for w in words:
40   x = w.replace(' ', '-')
41   if x in model:
42     results[w] = model.most_similar(positive=[x])
43   else:
44     unmatched.append(w)
45
46 def getitems(list):
47   mylist=[]
48   for a in list:
49     mylist.append(a[0])
50   return mylist
51
52 print ' *', 'unmatched:', unmatched
53 print ' *', 'matched', len(results), 'of', len(words)
54 for a,b in sorted(results.items()):
55   print("%s:%s" % (a,getitems(b)))
```

Listado A.8: MultiPerspectiveSummarizer.py (Interpretación y Plantillas)

```
########################################
#Estadisticas y analisis para plantillas#
########################################

#Filtra los topicos vacios para el analisis de datos
def cleanList(data):
  subselect=[]
  for a in data:
    if (a[3]!=""):#Filtramos los topicos vacios
      subselect.append(a)
  return subselect

#Devuelve un subconjunto de datos con unicamente los topicos relevantes o relacionados
def relevantes(data):
  relDict=getTopicDict.getTopicDict(config.RELEVANCIA_DIR+"ClusteredTopics.txt",REL_DIR) #Obtiene un diccionario de topicos relevantes
  final=[]
  for i in data:
    if(relDict.has_key(i[3])): #Existe como topic relevante o relacionado a uno relevante
      final.append(i)
  return final

#Devuelve un array con el sentimiento predominante en los comentarios (0=negativo, 1=positivo) y el porcentaje del sentimiento en el corpus
def opinionAnalyze(data):
  pos=[]
  neg=[]
  for a in data:
    if(a[2]=="positivo"):
      pos.append(a)
    else:
      neg.append(a)

  predom=""
  sent=0
  if(len(pos)>len(neg)):#Predomina positivo
    predom=pos
    sent=1
  else:#Predomina negativo
    predom=neg
  return [round(len(predom)*100.0/len(data),2),sent]

#Dado un sentimiento (negativo, positivo), devuelve el topico relevante con mayor inferencia en el corpus y su porcentaje de
```

```
    aparicion en el corpus
def topicAnalyze(data,sent):
  data=cleanList(data)
  data=relevantes(data)
  diccPos=defaultdict(lambda:0)
  diccNeg=defaultdict(lambda:0)

  for a in data:
    #print("%s:%s" % (a[3],a[2]))
    if (a[2]=="positivo"):
      diccPos[a[3]]+=1
    else:
      diccNeg[a[3]]+=1

  a=[] #El diccionario seleccionado en forma lista
  b={} #El otro diccionario
  if (sent=="positivo"):
    a=diccPos.items()
    b=diccNeg
  else:
    a=diccNeg.items()
    b=diccPos

  a=sorted(a,key=operator.itemgetter(1),reverse=True)

  #Seleccionamos uno con mas del 50% del corpus
  for i in range(len(a)):
    c=a[i]
    if(round(c[1]*100.0/(c[1]+b[c[0]]),2)>50.0):
      return [c[0],round(c[1]*100.0/(c[1]+b[c[0]]),2)]
  return []

#Devuelve una lista de los 3 topicos relevantes mas comentados y
    su inferencia en el conjunto de datos total
def mostCommented(data):
  limpia=cleanList(data)
  dic=defaultdict(lambda:0)
  limpia=relevantes(limpia)

  for a in limpia:
    dic[a[3]]+=1

  lista= dic.items()
  lista= sorted(lista, key=operator.itemgetter(1), reverse=True)

  final=[]
  for i in range(3):
    final.append([lista[i][0],round(lista[i][1]*100.0/len(data),2)
       ,len(data),len(lista)])
```

```
88
89    return final
90
91  #Analiza la contradiccion existente dentro del corpus para luego
        mostrarla en el resumen
92  def contraAnalytics(data):
93    data=sorted(data,key=operator.itemgetter(3)) #Ordenamos por
          topico
94    parcial=[]
95    topic=""
96    init=True
97
98    ######ANALISIS CONTRADICCION########
99    contradictorios=[]
100   topicos_dist=0.0
101   topicos_contr=0.0
102   ############################
103
104   for a in data:
105     if(init):
106       init=False
107       topic=a[3]
108     if(a[3]!=topic): #El topico ha cambiado y parcial tiene al
          menos un elemento
109       #Aqui se hace algo
110       positivas=[]
111       negativas=[]
112       for b in parcial: #Todos los elementos de parcial tendran el
              mismo topic
113         if b[2]== "positivo":
114           positivas.append(parcial.index(b))#Guardamos el indice
                de elemento en la lista parcial
115         else:
116           negativas.append(parcial.index(b))
117       sub=[]
118       #print("%s:%s:%s" % (len(positivas),len(negativas),topic))
119       topicos_dist+=1.0
120       if (len(positivas)>=1 and len(negativas)>=1):#introducimos
             un ejemplo de positivo y otro de negativo para cada
             topico
121         if(topic!=""): #Eliminamos deliberadamente los topicos
               vacios de la lista de ejemplos, sin emabrgo los
               contamos en las estadisticas, al ser tambien
               contradictorios.
122           #Del mismo modo nos quedamos unicamente con los topicos
                 que son relevantes o relecionados con ellos
123           relDict=getTopicDict.getTopicDict(config.RELEVANCIA_DIR+
                 "ClusteredTopics.txt",REL_DIR) #Obtiene un
                 diccionario de topicos relevantes
```

```
124             if(relDict.has_key(topic)): #Existe como topic relevante
                    o relacionado a uno relevante
125               contradictorios.append([topic,parcial[positivas
                      [0]][0],parcial[negativas[0]][0]])
126           topicos_contr+=1.0
127
128         #Se termina y se continua con el siguiente
129         parcial=[]
130         topic=a[3]
131         parcial.append(a)
132       else: #Se va rellenando parcial
133       parcial.append(a)
134
135     #Devolvemos el conjunto de topicos contradictorios y el
          porcentaje de topicos contradictorios que hay
136     return [contradictorios,round(topicos_contr*100.0/topicos_dist
          ,2)]
137
138   #Obtiene la media de los metadatos recolectados dados un hotel y
       un tipo de usuario
139   def metaAnalytics(hotel,tipo):
140     query="SELECT m.meta, m.metaScore FROM Metadato m LEFT JOIN
          Comentario c ON c.id=m.id_comentario WHERE c.hotel='%s' && c.
          userType='%s';" % (hotel,tipo)
141     result= run(query)
142     metaDict=defaultdict(lambda:0)
143     veces=defaultdict(lambda:0)
144     final=[]
145     for i in result:
146       metaDict[i[0]]+=i[1]
147       veces[i[0]]+=1
148     for i in metaDict.items():
149       final.append([i[0],i[1]/veces[i[0]]])
150     return final
151
152   ############
153   #PLANTILLAS#
154   ############
155
156   #Frase de Introduccion:
157   def fraseIntro(data,hotel,tipo):
158
159     tipo=tipo.lower()
160
161     if(tipo=="solo"):
162       tipo="solo travelers"
163     if(tipo=="friends"):
164       tipo="friend travelers"
165     if(tipo=="business"):
```

```
166       tipo="business travelers"
167
168     analizar= opinionAnalyze(data)
169
170     porcen=analizar[0]
171
172     opinion=""
173     sent=""
174
175     if(analizar[1]==0):#si es negativo
176       opinion="bad"
177       sent="negative"
178     else: #si es positivo
179       opinion="good"
180       sent="positive"
181
182     plantillas=["The %s who stayed at the %s had a generally %s
            opinion of this hotel, as the %s%% of the comments are %s." %
             (tipo,hotel,opinion,porcen,sent),
183     "At the %s, the %s that stayed there had mostly a %s opinion of
            their stay. We can say that because the %s%% of the existent
            comments are %s." % (hotel,tipo,opinion,porcen,sent),
184     "The %s has a %s reputation amongst %s, which is visible on the
            %s%% of the comments found on TripAdvisor that are %s." % (
            hotel,opinion,tipo,porcen,sent)]
185
186     seleccion = random.randrange(len(plantillas))
187
188     return plantillas[seleccion]
189
190   #Ejemplos:
191   def ejemplos(data,topic,sent):
192
193     subselect=[]
194     for a in data:
195       if (a[2]==sent and a[3]==topic):
196         subselect.append(a)
197
198
199
200     plantillas=["For example, you can see that on comments like \"%s
            \"." % (subselect[0][0]),
201       "An example of these comments is: \"%s\"." % (subselect[0][0])
              ,
202       "Proof of that are comments like \"%s\"." % (subselect[0][0]),
203       "This is evidenced by comments like \"%s\"." % (subselect
            [0][0])]
204
205     r1 = random.randrange(len(subselect))
```

```
206     r2 = random.randrange(len(subselect))
207 
208     if(len(subselect)>=2):
209       plantillas=["For example, you can see that on comments like
             \"%s\" or \"%s\"." % (subselect[r1][0],subselect[r2][0]),
210       "Examples of these comments were: \"%s\", \"%s\"." % (
             subselect[r1][0],subselect[r2][0]),
211       "Proof of that are comments like \"%s\" and \"%s\"." % (
             subselect[r1][0],subselect[r2][0]),
212       "This is visible on comments like \"%s\" and \"%s\"." % (
             subselect[r1][0],subselect[r2][0])]
213 
214 
215     seleccion = random.randrange(len(plantillas))
216 
217     return plantillas[seleccion]
218 
219   #Les gusta:
220   def likes(data):
221 
222     a= topicAnalyze(data,"positivo")
223     if (len(a)==0):
224       return ""
225     plantillas=["They talk positively about \"%s\" as the %s%% of
             the comments about this topic are positive." % (a[0],a[1]),
226     "The users seem comfortable talking about \"%s\" because they
             express positiveness on the %s%% of the comments related to
             this topic." % (a[0],a[1]),
227     "The customers tend to be positive when talking about \"%s\"
             topic, which is visible on the %s%% of the comments for this
             topic." % (a[0],a[1])]
228 
229     seleccion = random.randrange(len(plantillas))
230 
231     return plantillas[seleccion]+" "+ejemplos(data,a[0],"positivo")
232 
233   #Les desagrada:
234   def dislikes(data):
235     a= topicAnalyze(data,"negativo")
236     if (len(a)==0):
237       return ""
238     plantillas=["On the other hand the users seem to dislike talking
              about \"%s\" as we can see negative feelings on the %s%% of
             the comments related to it." % (a[0],a[1]),
239     "On the other hand the customers tend to be negative when
             talking about \"%s\" as you can see on the %s%% of the
             comments." % (a[0],a[1])]
240 
241     seleccion = random.randrange(len(plantillas))
```

```
242
243     return plantillas[seleccion]+" "+ejemplos(data,a[0],"negativo")
244
245   #Frases intermedias:
246   def intermedias(data):
247     datos=mostCommented(data)
248     plantillas=["We can also see that, from the %s reviews crawled
          for this type of traveler and the %s topics detected, the
          most commented aspects for this hotel were the \"%s\",
          noticeable on the %s%% of the comments, \"%s\" with a %s%% of
           appearance, and \"%s\" with a %s%%." % (datos[0][2],datos
          [0][3],datos[0][0],datos[0][1],datos[1][0],datos[1][1],datos
          [2][0],datos[2][1])]
249     seleccion = random.randrange(len(plantillas))
250     return plantillas[seleccion]
251
252   #Ejemplos diversidad de comentarios:
253   def ejemploDiversidad(data,topicSelect):
254     plantillas=["For the topic \"%s\" they comment \"%s\" and also
          \"%s\". The same occurs for the %s%% of the previously
          commented topics on this hotel, so you cannot establish a
          definitive conclusion about them." % (data[0][topicSelect
          ][0],data[0][topicSelect][1],data[0][topicSelect][2],data[1])
          ]
255     seleccion = random.randrange(len(plantillas))
256     return plantillas[seleccion]
257
258   #Sobre la diversidad de opinion de un viajero:
259   def diversidad(data):
260     res=contraAnalytics(data)
261     r1=r2=r3=0
262     plantillas=[""]
263     if(len(res[0])==1):
264       plantillas=["This type of traveller has shown different
            opinions about the topic \"%s\"." % (res[0][0][0]),
265       "The diversity in opinion in this type of traveler is shown on
             the topic \"%s\"." % (res[0][0][0])]
266     elif(len(res[0])==2):
267       plantillas=["This type of traveller has shown different
            opinions about some topics like \"%s\" and \"%s\"." % (res
            [0][0][0],res[0][1][0]),
268       "The diversity in opinion in this type of traveler is shown on
             topics like \"%s\" and \"%s\"." % (res[0][0][0],res
            [0][1][0])]
269     elif(len(res[0])>=3):
270       while(r1==r2 or r1==r3 or r2==r3):
271         r1=random.randrange(len(res[0]))
272         r2=random.randrange(len(res[0]))
273         r3=random.randrange(len(res[0]))
```

```
274       plantillas=["This type of traveller has shown different
              opinions about some topics like \"%s\",\"%s\" and \"%s\"."
              % (res[0][r1][0],res[0][r2][0],res[0][r3][0]),
275       "The diversity in opinion in this type of traveler is shown on
               topics like \"%s\", \"%s\" and \"%s\"." % (res[0][r1][0],
              res[0][r2][0],res[0][r3][0])]
276     seleccion = random.randrange(len(plantillas))
277     if(len(res[0])<1):
278       return ""
279     if(len(res[0])==1):
280       return plantillas[seleccion] +" "+ ejemploDiversidad(res,0)
281     elif(len(res[0])==2):
282       return plantillas[seleccion] +" "+ ejemploDiversidad(res,
              random.choice([0,1]))
283     else:
284       return plantillas[seleccion] +" "+ ejemploDiversidad(res,
              random.choice([r1,r2,r3]))
285
286   #Sobre los metadatos:
287   def metadatos(hotel,tipo):
288     res=metaAnalytics(hotel,tipo)
289     plantillas=["Finally, we list below the average scores of some
          especific aspects of the hotel that have been ranked by the
          users:"]
290     seleccion = random.randrange(len(plantillas))
291     final= plantillas[seleccion] + "\n"
292     for i in res:
293     final+= "%s: %s/5\n" % (i[0],i[1])
294     return final
```